Malaysia M $ 9.45 Sweden 30:-SEK $2.75 USA

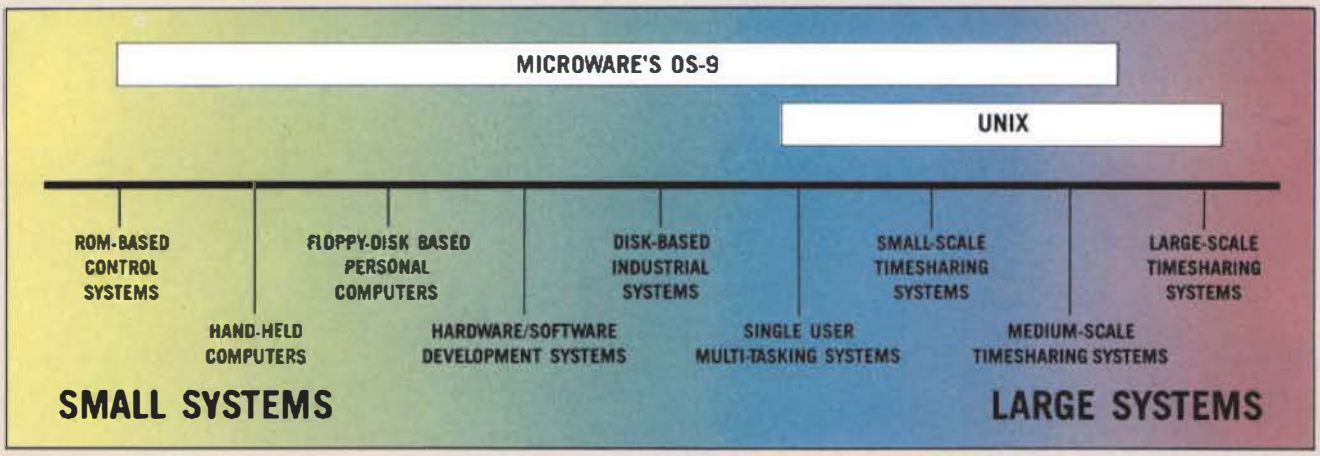
68 MICRO JOURNAL

VOLUME VIII ISSUE IV • Devoted to the 68XX User • April 1986

"Small Computers Doing Big Things"

0 74470 12810 04

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTOR

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIEWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
1-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
Sct Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

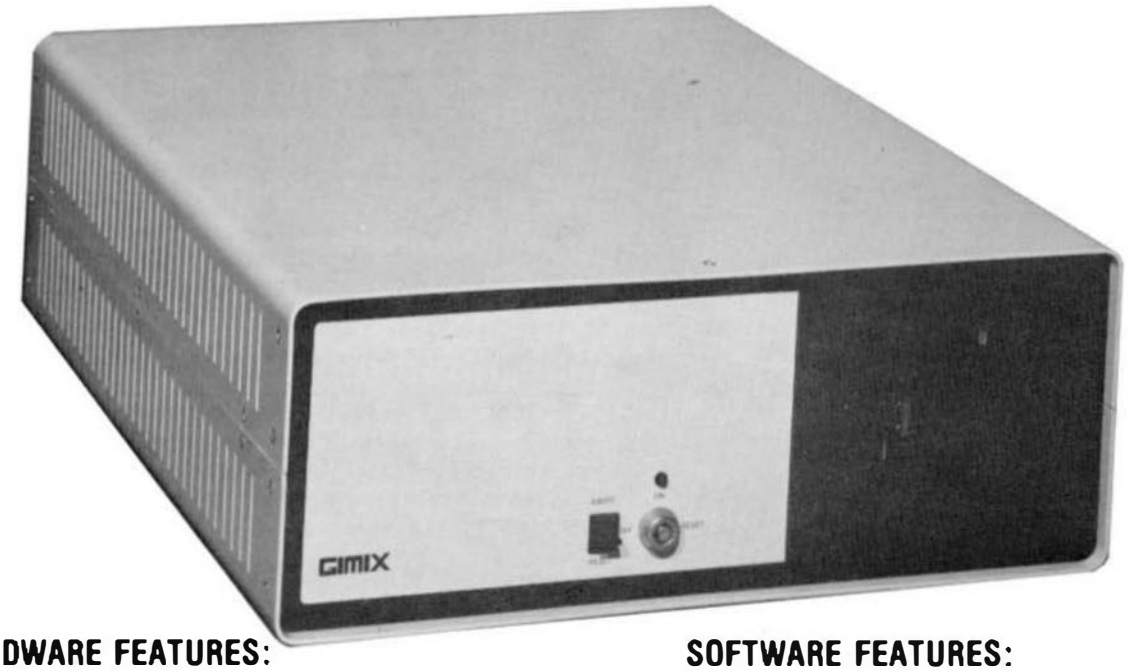
SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL, GMBH
Porphystrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 65025

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055

'68'

MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS – HARDWARE

Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S.9 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Contents

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year.** Must be in **U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format with **3.5 inch column width.** All disks will be returned. Articles submitted **on paper** should be **4.5 inches in width** (including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: **.sp** space, **.pp** paragraph, **.fi** fill and **.nf** no fill. Also please **do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles
OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microproceasor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGOC1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files Included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.90 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG -020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the **REAL KICK!**"
Only from DATA-COMP

This is the NCC, world beater CMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your apecs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be Turtle or Thoroughbred?

MUSTANG-020 System Prices
Effective November 1985

Mustang-020 SBC, wired & tested with 4 DB25 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives.....$2750.00

M020 Cabinet and P/S, for Mustang-020, less cables..... $269.95

M020 Cables, dual floppy or winchester, specify which - floppy or winchester.....$39.95

M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies.....$79.95

M020F Floppy, 80 track, DD/DS.....$269.95

OS-9, SPECIAL Mustang-020 version.....$350.00

MC68881F/P co-proceasor.....$495.00

10 Megabyte Winchester.....$695.00

20 Megabyte Winchester.....$895.00

Winchester Controller (Xebec-).....$395.00

Note: for orders of complete systems (Mustang-020, cabinet & P/S, disk drives and OS-9, deduct 5% from total package. (limited time offer) See opposite page.

**** Special Winchester Notice ****

The Mustang-020 device descriptors will allow you to use practically **ANY** winchester drive supported by XEBEC controllers.

* * *

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H. Complete System include $20.00. All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

Dealer & Quantity Discounts Available

* Mustang-020 is trademark of Data-Comp-CPI

MUSTANG-020 Super SBC™

Mustang 020 Features

- 12.5 MHz MC68020 full 32-Bit wide path Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatible with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
- Optional 68881 Floating point Coprocessor (12.5 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
- 2 Megabytes of RAM (512K x 32-bit organization)
- Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
- 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
- Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
- Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
- Time-of-Day Clock/Calendar w/battery backup
- Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
- Mounts Directly to a Standard 5 1/4" Disk Drive
- SASI Interface for Intelligent Hard Disk Controllers
- Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
- 5-bit sense switch, readable by the processor
- Hardware single-step capability

MUSTANG-020 Benchmarks **
Time Seconds

Type System	32 bit Int. Loop	Register Long Loop
IBM AT 7300 Xenix Sys 3	9.7	No Registers
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " " "	5.1	3.2
68000g OS9 68K 8 Mhz	18.0	9.0
68000 " " 10 Mhz	6.5	4.0
MUSTANG-020 68020 MC68881 OS9 16 Mhz	2.2	0.88
MUSTANG-020 68020 MC68881 UNIFLEX "	1.8	1.22

** Loop:
```
Main()
{
  register long i;
  for (i=0; i < 999999; ++i);
}
```

Estimated MIPS - MUSTANG-020 - 2.5 MIPS
Motorola Specs: Burst up to 7 - 8 MIPS - 16 Mhz

☎ **(615)842-4600**
Telex 5106006630

For a limited time we will offer $400 Trade-In on your old Q--- 68008 or 68000 SBC, must be working properly and complete with all software and cables. Call for more information!

**** ACTION PROVEN ****

Mustang-020 Software

OS-9

OS-9	$350.00
Basic09	300.00
C Compiler	400.00
Fortran 77	*400.00
Pascal Compiler	400.00
DUNCASOFT-PASCAL	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
SCULPTOR+	*Call
COM	*Call

** See discount below

UniFLEX

UniFLEX	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
Fortran 77	450.00
SCULPTOR+	*Call

** See discount below

* New Items

Standard system shipped 12.5 MHz
Add for 16 Mhz..68020....$400.00
Add for 16 Mhz..68881....$400.00
8 Port expansion board use two total of 20 RS232 ports wired & Tested.....each....$498.00

SCULPTOR+..We are USA distributors for SCULPTOR+. Call or write for site license or multiple discounts.

**** Software Discounts**

Call for software discounts from 10-70% for buyers of these systems, from Data-Comp. Limited offer. Call!

The MUSTANG-020 is already on the job! And winning aclaim in industry, commerce, business and several government agencies. The delivery times were close to schedule. We are hearing back nothing but praise (and more orders).

If you are considering the purchase of a Mustang-020, be advised that the price will increase in the second quarter of this year.

Experienced users are awed at the tremendous power and speed of the Mustang-020, from Data-Comp. Especially when compared with other 68XXX systems. Not only is it more practical than all the others, but it is much more cost efficient. Compare it to any other 68XXX and you will see why!

	Dual 5" 80 trk. Floppy No Winchester		Winchester & 1 Floppy
020 Board	$2750.00		$2750.00
Cabinet	269.95		269.95
5"-80 trk Floppy(2)	539.90	(1)	269.95
Floppy cable	39.95		39.95
OS-9 68K	350.00		350.00
		Winchester cable	39.95
Total System	$3949.80	Winchester controller	395.00
Less 5%	-197.49	10 MegByte Winchester	695.00
	$3752.31	Total System	$4809.80
S/H UPS	20.00	Less 5%	-240.49
			$4569.31
Total	$3772.31	S/H UPS	20.00
		Total	$4589.31
		With 20 MegByte Winchester Add:	200.00
			$4789.31

NOTE: 68881 Co-Processor Add $495.00
UniFLEX $450.00
less $350.00 (OS-9) Add $100.00

Prices and Specifications subject to change.

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

Standards

The first sales of PAT have begun. It was not long after this event, that I received a letter from a purchaser. He had just bought a new terminal that uses the ANSI standard terminal control codes. Pat won't work with that terminal, nor can it be configured by the user to make it work. (Read on here a little please). I've prepared a special version of PAT to work with the ANSI codes, and still be configurable by the user, and I have sent a copy to that customer, so he is, or will soon be in business with PAT.

Leave it to a committee to foul things up. My favorite story about committees is the one that goes something like the following: There once was a committed that set out to design a horse. The result of course was the Camel. This ANSI standard is certainly a Camel. Standards committees seem to obey the following pair of rules:

1. If the manufacturers of the device have been using a more or less de-facto standard, be sure that the new standard is entirely different so as to obsolete all devices produced before the establishment of the standard.

2. Be absolutely certain to make the standard way more cumbersome and less efficient to use than the de-facto standard.

The ANSI standard meets both of those goals admirably. Until the standard came along, nearly every "run of the mill" terminal used a four character sequence to position the cursor on the screen. The sequence ESC , Lead-in Character, Line, Column has generally been used with a few variations. The lead-in character is generally some printable character. Many terminal manufacturers use the ASCII code for "=", though there are others. The code for line and column generally use the binary byte value that represents that number, usually, but not always with the value 32 added to them. (The addition of 32 to the line or column value simply gets them out of the range of ASCII control characters and into the printable character range). Thus four characters are used to put the cursor somewhere on the screen. What could be simpler for a software supplier. Let the user specify the two initial characters, the escape and whatever the lead-in is, and then let him specify the "offset" to be added to the line or column number. One slight complication arises in that some of the terminals (just a few) want the column before the line. OK, ask the user which comes first and set a variable value on that basis, and let the program decide by the variable value which to send first.

Now along comes a committee and decides to make the cursor positioning string anywhere from 6 to 9 characters long. (Remember that some of the new terminals can display 132 columns, requiring a three digit column number). In fact the standard is:

ESC [LL ; CC H

> This ANSI standard is certainly a Camel.
>
> Leave it to a committee to foul things up.

Those are all ASCII characters, (The control string has no spaces - I've added them for clarity) LL standing for the line number as a one or two digit ascii value. In other words if you want to put the cursor on line 12, you send the ASCII code for 1 followed by the ASCII code for 2. Similarly the value CC is the ASCII code for the number representing the column. 69 is sent as ASCII 6 followed by ASCII 9 (HEX codes 36 39). That means several things to the software supplier. First of all, there are several "separator" characters that must always be sent. Secondly, if the line is less than 10, only one ASCII character need be sent. That is, the string can be shorter by a character if the line or column is less than 10. By some sort of reasoning that I don't understand, apparently the committee decided that the separators [, ;, and H would be used so that the line and column could be only one character each under some circumstances. If they had made the line two characters always (use the leading zero), and the column three characters, the string could always be 7 characters long, namely ESC [LLCCC. They have further simplified things by making a value of either 0 or 1 refer to the first line or column. In other words, in terms of the default, they have added an offset of 1 to the line and column numbers.

I suppose there is some sort of rationale for making the numbers decimal ASCII values, but certainly converting a byte representation of column 35 (00100011) to $33,$35 takes some little processor time. If the product is a screen editor that updates the column number on the screen with every character typed, the overhead is some 1600%. That is to type one character to the screen requires 16 more characters. Eight of them position the cursor to the place for the column number (actually then the column number is output), and eight more characters put the cursor back where the next character is to be typed on the screen. Can you imagine running a screen editor on such a terminal via modem at even 1200 baud?

The old default four character sequence is bad enough when running software that moves the curaor around the screen. Why make it twice aa bad? Such is progreaa! Deliver me from ANSI standard terminals. I hope at least some of the manufacturers are smart enough not to adopt that standard!

Mixed Mode a Mixed Bag

We (at the company where I spend my workday) have been using PL/9 to write programs for about three years now. In a few instances when the program didn't work as expected, We've changed statements around a little and gotten the program to work properly without actually diacovering what was wrong. The other day an instance of the mysterioua problem occurred, and this time I think we figured out what was going on. PL/9 baa some rather simple rules with regard to handling what is called mixed mode arithmetic. That is, the situation where you are performing arithmetic with mixed variable types as for example, multiplying a REAL variable by an Integer variable. The simple caae ia that in an assignment atatement, all the variables and constants in the expression are converted to the type of the variable to which the result of evaluating the expression is to be assigned. That is, if you have the situation REAL = INTEGER * REAL; The integer will be converted to a REAL, the multiplication done, and the REAL result assigned to the REAL variable on the left of the statement.

That is all reasonable. If you have the reverse aituation, witb the INTEGER at the left, and a mixed expression on the right, all REALa in the expreaaion will be converted to INTEGER first. If you don't want that to happen, you enclose the expression in parentheses and use the intrinsic function FIX, which allows the calculation to be carried out in REALa, the result converted to an INTEGER and assigned to the variable at the left of the equal sign.

Now, what happens in a condition for an IF THEN, a WHILE, or a REPEAT UNTIL? Well, we had assumed that the variables would always be "promoted" to the type of the "higheat" type in the expreaaion. To our surprise we found that the variables are all converted to the type of the FIRST variable in the expression. If PI has been defined as the REAL number 3.14159265.... you would expect IF 2 * PI > 6.1 to evaluate TRUE. However it does not. 2 is an integer ao PI gets changed to an integer of value 3. 2*3 is not greater than 6.1 ao the expression evaluates false. The aame "error" happens whenever an integer or byte variable ia first in an expreaaion containing a REAL variable. Now turn the expression around to read IF PI * 2 > 6.1 and it DOES evaluate TRUE since the 2 is converted to a REAL 2.0 and the multiplication yielda 6.2832.... which ia certainly greater than 6.1.

While this is not really a BUG as auch, but rather a "tbat'a how it works" item, I could find no reference in the PL/9 manual to mixed mode arithmetic in a condition. Nowhere does the PL/9 manual state that the variables in an expression in a condition will be converted to the type of the first variable in the expression. Now that we know that it works that way, we will be careful to put the REAL first when we do aomething like that. Even in our real world machine applications, the situation doesn't arise very often. Usually the condition in an IF THEN is a logical condition like End of File or the termination of a count uaing byte or integer variables. I've already written Windruah about the oversight of not mentioning thia little poaaible problem in the manual, but I thought I would mention it here for the benefit of the PL/9 uaera who read this column.

I haaten to add that thia is in no way a complaint about PL/9 overall. It ia atill my preferred language for applications involving hardware interfaces. Its output code is very efficient and the faat single paas compile operation aaves a great deal of time that would otherwiae be apent waiting for the program to compile for the 389tb time. I just wanted to point out that the way that PL/9 handles expressions in a condition could cause mysteriaua problems if not understood.

Evaluating Compilers

I recently received two more entries in the 6809 FLEX aoftware area for reviaw. That fact led me to think about how I sort these compilers out in order of desirability. I thought I might spend aome time here diacuaaing each factor that enters into the process of the evaluation. Let me preface this by saying that I am prejudiced by the fact that I must use a compiler to generate code that will be installed in a stand alone microproceaaor system in a product. If I were using the compiler for strictly personal projects, I might have a little different outlook. If I intended to produce commercial software with it, I might have a third point of view.

Language

Por me personally, the language doesn't enter mucb into the picture except that it muat be possible to write a program that can be read a year later by me or next week by someone elae, and be understood so that it can be maintained for a customer. (I mean maintained even if I leave the company due to my choice, accident, or illness). This constraint requires that the language be one of the so called "structured languages" and rules out the "threaded code" languages. Many languages that don't force structured code allow it, and these are OK if not the beat to use.

Compiler Features

I am now talking about the compiler program itself, quite aside from the language. Does the compiler allow me to liat the program? Can I compile it without creating an output file ao I can aee if there will be any errors that the compiler can catch? How good is ita error trapping? (I recently looked at a compiler that let me compile a program containing a jump to a non-existent label and reported an error only when the compiled program was run. Unresolved references should be caught by the compiler.)

Time to Compile a Program

I have worked with compilers that, given a 15 page program would take anywhere from 30 aeconda to 30 minutes to compile it. Of course I mean different compilers take different amounta of time to do their job on the equivalent program. Particularly in my work applications, I can't afford to wait 30 minutes for my program. Only the compilers that run in the range of three minutes or better for a program of that aize are usable. I should add that aome compilera seem to execute in a more or leas fixed time, almost independent of the program length, while otherа are very sensitive to the length of the program. The languages that selectively link in library routines from a library package tend to be more constant in compile time. They have to read their 60 to 100 aector library file completely regardless of how many library routines they copy out to the output file.

Output Code Size

My recent teats of a couple of new compilera brought about a new teat program. It would aeem reaaonable to aee how small a program can be compiled efficiently. Hence my new teat program that aimply prints "Hello There" to the terminal. The two recent compilers acored

6800 bytes for one of them and 600 for the second. PL/9 (my standard of comparison) compiled 90 bytes for that program.

Don't jump to conclusions on that one test, however. It is no better than a single "benchmark" for evaluating a compiler's output code execution speed. Some compilers have a large package of runtime routines that are included in the output code, but then they generate smaller incremental code per line of user program than others.

A prime example of this sort of a compiler is Lucidata Pascal. That Pascal implementation is a P-Code compiler. That is, the compiler translates the input source code into an intermediate code that is easy to interpret, and the user program is then "bound" to a portion of the interpreter, hopefully only that portion necessary to run the program. Lucidata Pascal has a reasonably small interpreter, and is very efficient at generating the incremental code for the user program. Therefore, though it looks inefficient for a very small program, a compiler that has a very small runtime package but generates code less efficiently soon catches up with it in program size. You must therefore test small programs and large ones in order to get a feel for the compiler.

In the present case, I wrote a square root routine for the compiler that generated the very large code for "Hello There". My simple program to input a number and find its square root generated 10,500 bytes of code and it took 1.64 seconds to find the square root of 3, though it did so to 9 digits precision in that time.

I coded that same program in PL/9 and it generated 1213 bytes of code, and found the square root of 3 in 0.070 seconds. I wrote the supplier of the new compiler that a new compiler that generates 8 or 9 times as much code and executes the same program 23 times slower than an already available one doesn't have much going for it.

Hardware Accessibility

This and the next aspect of compilers are probably more important to me as an implementor of programs that run hardware, than some others. If the language or the compiler implementation allows the user to specify a variable AT a memory location, it is easy to access hardware serial and parallel ports directly. One simply declares a BYTE or INTEGER variable at the address of the port and then either assigns values to the port to write to it, or assigns its value to another variable to read from it. Pascal and PL/9 have this feature. C uses another method that is equally useful. You simply create a pointer and point it at the port address. You can then reference the port as a memory location via the pointer and read from or write to it.

Assembler Code Interface

If you are really concerned with speed, or have some fairly complex hardware attached to the computer so that you want to use drivers written in assembler, it is nice if the compiler allows you to write procedures or functions in assembler and connect them to the compiler. Alternately, you may be able to embed assembler code in the compiler source code.

Portability

For my work projects, this doesn't enter into the picture greatly, but for most people, including software suppliers, it is very important. PL/9, though my overwhelming choice for most every other reason, is a special language written for the 6809. Programs written in it are simply not portable to other processors or operating systems at this time. (A 68000 cross compiler is in the mill, so that PL/9 programs may soon be portable to 68000 systems.) The very best language for portability is "C", and it seems that the implementors have realized that fact and made their compilers VERY standard.

> **Thanks to the foresight of Kernighan and Ritchie in keeping to the "de facto" standard library, if I write a program that runs in "C" on one computer, it is very likely .... to run on another computer.**

Notwithstanding the words of Bud Pass in his column about the various versions of "C" that run under FLEX, and how they don't all handle everything identically, "C" is head and shoulders above Pascal for portability. The main reason is that though "C" doesn't have any I/O functions in the "base language" it has a standard library, usually written in "C" that provides all the I/O functions including a choice of low level disk interface or high level disk interface.

Thanks to the foresight of Kernighan and Ritche in spelling out these interfaces, and to the implementors in keeping to the "de facto" standard library, if I write a program that runs in "C" on one computer, it is very likely to require only minor modifications to run on another. I speak from the experience of having translated my JUST program to "C". It is about 1000 lines of code, and the difference between the McCosh version that runs under FLEX and the Lattice "C" version that runs under MS-DOS is all in 5 lines of code, dealing with determining whether output is going to the printer or the terminal. In addition, one constant is declared to be $0D in the FLEX version and $0A in the MS-DOS version because the two operating systems have different "standard" text files. Flex text files use CR to end a line and MS-DOS uses LF.

The problem with Pascal in this regard is that file operations are not even defined in the language standard. This is not a criticism of the language, which was originally developed to be run on large mainframe machines to teach students how to write programs. The lack of a standard or an example, however, has led to every implementor of Pascal handling the opening and closing of files in a different manner. Some Pascal implementations get arguments from the command line, and some do not. With the first, you can include a filename on the command line that runs the Pascal program. With the others, you have to do it the BASIC way, and prompt for a filename from within the program.

Source Code Readability

I touched on this in the introductory remarks. Readability or "self documentation" is desirable when code has to be maintained over a long period and by more than one person. The languages that are considered to be "structured" such as Pascal and C (and many of the specialty languages like PL/9 and Whimsical), produce code

that is more readable than many of the other languages. In case I've lost anyone here, structured programming is programming using techniques of loop control that avoid using GOTO or its equivalent. Structures like DO WHILE, REPEAT UNTIL, and FOR, all serve to clarify the extent of loops, and show the structure of the program better. Things like Pascal's CASE statement or the equivalent SWITCH statement in C, also are much clearer than a long string of nested ELSE statements.

The requirement to "declare" variables and to specify their type (Integer, Byte, Character, Real, Boolean, etc) is a great aid to someone reading someone else's program. The use of meaningful variable names is of course a matter for the programmer, but some languages only allow two or three character names (mostly BASIC) which make it impossible to make them very meaningful.

All of the "structured" programming languages have provision for compound statements, a group of statements surrounded by some sort of bracketing words (BEGIN-END, IF ENDIF, or symbols such as {}. Once you have used these niceties it is hard to program in a language that doesn't have them. (Maybe rather than hard, I should say awkward or uncomfortable).

Of course comments are the programmer's prerogative, but a language like Pascal or C requires far fewer comments to explain it than a language that has less loop control structures and requires frequent use of GOTO.

Language "Verbosity"

No, I'm not trying to be funny. Verbosity is my term for the "wordiness" of the source code. Being calculation oriented, I think the height of verbosity is found in COBOL. ADD 2 TO A GIVING C seems a little excessive when in most languages the statement looks more like C=A+2. Verbosity and readability are somewhat connected. You might prefer the "index+=2;" of "C" to the INDEX = INDEX + 2; of other languages, but the second form is probably more readable at least until you get used to the shorthand of "C". The curly braces {} of "C" which replace BEGIN and END, on the other hand might appeal to you as simply producing a less cluttered looking program.

Execution Speed

I've saved this one for near the end of the discussion because I think it is usually overemphasized in the testing of compilers. To discredit a compiler that scores well in all the other areas simply because its output code executes half as fast as the fastest one might be very foolish, particularly if the application doesn't demand blinding speed! Any compiler whose output code executes fast enough for the application, is worth considering. I note that as a general rule, the compilers that compile smaller output code run faster as well.

Documentation

I mean here, the instructions that come with the compiler. Documentation can range from several pages to several hundred pages. The largest and best documentation I have ever seen is the two-volume set that accompanies PL/9. It is both a technical reference and a tutorial on using the language. Many of the Standard Language compilers simply state outright that the manual is not a guide to learning the language, but contains instructions on using the compiler itself. There are many excellent books on Pascal and "C", and the manuals that accompany those compilers usually contain the author's recommendations concerning books that will teach you the language.

Some documentation is so-so, the authors using "cute" variable names and peculiar examples that do nothing but confuse the reader. The author is usually so familiar with his compiler or the syntax of the language he has invented (in the case of a non-standard language) that his examples lack clear explanations, or are missing altogether. I tend not to worry terribly about documentation, being sort of adventurous and willing to try, and fail a few times before becoming discouraged.

Whatever you do when that new compiler arrives, don't sit down and read the manual from cover to cover, word for word, four times. You'll become discouraged. You don't have to know every possible feature of it to start using it. Skim through the manual briefly. Read the part on how to use the compiler. Then type in the usual test program of a few lines and compile and run it. Now gradually expand your test program to try various features of the compiler or language. If you don't need certain functions for the sort of things you normally program, don't bother learning them. If you ever do need them, the manual will be there for you to read.

Compiler Size

This is usually but not always related to the time to compile a program. Some of the compilers, (for some reason the "C" compilers in particular) are very large, occupying some 600 disk sectors with all their various parts. Obviously if you have a Single Sided, Single Density 5 1/4" disk drive, the compiler requires two disks, which may not be practical if there is not a convenient way to split it into a couple of operations. Even if a compiler of this size is very fast in execution, you still have to read all 600 sectors every time you compile a program. The disk access time to read the compiler and runtime library might be the largest part of the time to compile a program.

*** END ***

More Feedback

Everett Greene who has corresponded with me several times in the past, has written to dispute my claim that higher level languages result in much less source code than assembler does. He didn't disagree with the claim that sometimes or even usually the high level language source is shorter than the assembler source, but he found my 5 to 10 times shorter claim to be excessive. He did say that perhaps such ratios could be reached in very calculation intensive programs.

Well, Everett, I have a 12 page calculation in Assembler that reduces to just about 12 lines in a high level language. That is approximately a 50 to 1 reduction. I will grant, however, that some programs, particularly short ones can be almost the same number of lines. Just about anything I've tried has however been at least reduced to half the lines when done in a high level language. Perhaps 5 to 10 times is too high for an "average" program for you, but it is not, for the types of programs that I do.

Wrap Up

Well, I've finally done it. This month's column is too big to edit in a single "chunk". I will have to break it off very soon.

I just have to mention one more thing. The company has ordered a "Mustang-020" computer from Data Comp. I am anxiously waiting for it with a copy of Microware "C" for it in hand. The first order of business will be to finish a version of PAT in "C" for it. I have started the conversion process but the Mustang will speed up the process greatly.

Basically OS-9

Ron Voigts

ONE DOWN AND MANY MORE TO GO!

This is an exciting month! This month marks the one year anniversary of the BASIC OS-9 column. Last year in April, the column made its debut in the 68' Micro Journal. It was an offspring of a previous column that appeared in Color Micro Journal. I learned from writing that column that it was impossible to write one specific area without getting involved in the entire OS-9 spectrum. From talking and listening to other OS-9 users, I feel there is a need for useful information and programs.

The columns have included using the commands, understanding the system and knowing the languages that run on OS-9. Hopefully there is something for everyone. The monthly programs were in Pascal, Basic09, C and Assembly Language. They include utilities, general purpose and fun programs. Maybe sometime in the future we'll be able to offer them on disk for those who don't like to do all that typing.

Some of fun things have been the reviews for the different OS-9 products out. I have received letters from readers. I even had one reader show up at the door to "shake my hand." All in all, I would say it has been an exciting year. But so much for my sentimental ramblings. Onward to the future and full speed ahead!

MEMORIES

This month I was planning on talking about memory and the OS-9 module. RAM, as it is sometimes called, permits the computer to serve the operator. It makes the computer a versatile and flexible tool. In recent times the amount of RAM one has, has become synonymous with the power of the system. I overheard one computer enthusiast remark that he was limited by the amount of RAM he had. Just 16K more memory was all he needed. How we use memory is important. In OS-9 memory is managed by the "kernel".

The kernal is at the heart of OS-9. Or perhaps it is more appropriately called, the brain. It has a number of extremely important jobs. It acts as the administrator, supervisor, and resource manager for the system. It takes care of system initialization. It processes interrupts and service requests. The kernel also manages multiprogramming and memory. (As a side note, if you enter MDIR, you won't see its name in the module directory. There are two modules that you will see. They are OS9P1 and OS9P2. They make up the kernel.)

Memory management is important to the system. The physical assignment of memory must be taken care of. It differs between Level I and Level II systems. On Level I, all memory is contained in 64K of RAM. OS-9 and the users share a common memory space. On Level II, OS-9 and each user is assigned a private memory space. The size of it may contain up to 64K bytes, depending on the hardware configuration. All users may share a module, but RAM for data is assigned in each user's memory map. In either case, memory modules must be assigned to locations. Data areas are assigned to them. Their names go into the module directory. The kernel handles all of this, based on information in the memory modules format.

The memory module is the only thing that gets loaded into memory. The module has a specific format that passes information to the system. It's general format looks like:

Relative Address	Use
$0000	Module header
$0009	Execution Offset
$000B	Permanent Storage Size
	Module's Body
	CRC Check Value

The module header contains information about the module itself. It tells the module's size and name. It contains the module type and language. It gives its attributes and revision. I'll tell more about it later.

The execution offset is the relative start address from the first byte of the module. This is where the program is to start execution. Some modules may have multiple entry points. For example a device driver could have a table that looks like:

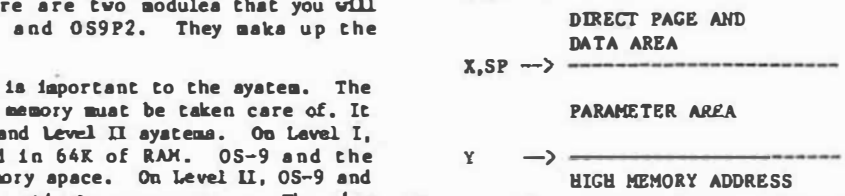
```
START  lbra INIT   initialize drive
       lbra READ   read from device
       lbra WRITE  write to device
       lbra GETSTA get status
       lbra PUTSTA set status
       lbra TERM   terminate device
```

This table offers 6 points of entry in it.

The permanent storage size is the amount of memory the module needs to run. It is allocated at the time the module is created. How it is allocated depends on the way the module is created. An assembly language program may declare some variables like:

```
A       rmb     2
B       rmb     2
NAME    rmb     10
```

This says that our data area must be have at least 4 bytes for two integers and 10 bytes for a string. At run time, a data area will be assigned to the program. The standard configuration is:



These registers are acting as pointers to the data area. The integer, A, will be at the location pointed to by U and DP. B will be 2 bytes past U or location U+2. NAME will be at U+12. The X register and Stack Pointer point to the bottom of the data area. Whenever anything is

pushed onto the atack the pointer movea upward toward the data area. Hence the OS-9 aaaembler manual's warning to supply a generous atack area for your programs. The parameter area contains the parameters passed with an OS-9 command.

The nice thing about OS-9 is that it can change the data area size at run time. To the end of a command can be added:

```
#nK  or #m
```

where n is the amount of memory desired or m is the number of pages. The data area will be dynamically increased. Commands like COPY or BACKUP will use the extra memory you give it. It will improve their efficiency and speed.

The OS-9 EDIT module is a command that you use extra memory, if you want. Normally it uses about 4K of memory, but you can ask for more. Alao, you can get more memory from it after it is running, by using its M command. Entering a:

```
E:M10000
```

will cause EDIT to increase its data area size by getting more memory from the system. Another solution is to alter the module EDIT so that it immediately gets the larger data size. To do this try the sequence:

```
OS9:rename edit edit.bak
OS9:load edit.bak
OS9:debug

Interactive Debugger
DB: l edit
    9900 87
DB: . .+0b
    9909 0C
DB: =27
    9909 27
DB: q

OS9:save edit.temp edit
OS9:verify u <edit.temp >edit
```

You now have an EDIT that has over twice as mucb buffer apace. What we did was load EDIT into memory, use DEBUG to change the byte at relative address $09 from $0C to $27, SAVEd a temporary version and used the "u" option of VERIFY to correct its CRC and make a new file. If everything works you can delete the backup and temporary files.

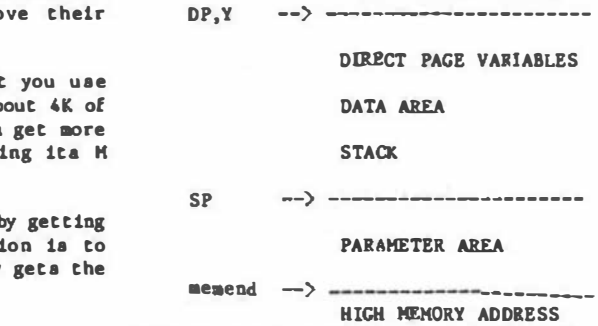
The modulm body is where the program code goes. It may I-Code generated by Basic09. Or perhaps it is P-Code from the Pascal Compiler. It may be machine code created by an assembler or from C source code. The information may not even be executable like the device descriptor.

Last is the CRC number. The cyclical redundancy check is a special 3 byte number that comes at the end of the module. It is calculated starting with the first byte and continuing up to the CRC number, the last three bytes. It is used to check if a module is intact.

DIRECTORY ALPHABETIZER REVISTED

Last November in the column, I presented a C program called DALPHA. The program would alphabetize OS-9 directories. Things did not go smoothly for many. I received a letter from Wayne Setzer of Charlotte, NC. Wayne writes that at best by reducing the LENGTH of the directory storage to 50 entries, he gets a stack overflow. When it is set to the LENGTH in the program, 150 entriee, the system hangs up. David Lynde's letter in the January 68 MJ, pointe out "When running on a large directory the program hung up completely...any program that bas more than 1000 bytes of 'auto' variablea would encounter this problem." Well there is a problem. I'm guilty! I feel like the chef who gave out a recipe, but didn't tell bow to cook it. In this caae, the program is correct, but the method to compile it needs revealing.

Let's talk a little about C programs. When you write a progrsm, it gets titled "main()". The C compiler turns it into a linkable module. The "mainline module" is cstart.r. It sets up the variable area. It prepares argv and argc. It also adjuste the Y register to point the start of memory. Finally a "lbsr main" is made and your program is running. C.link combines everything and makes them into a aingle memory module. The memory map for a C program looks like:



This an overly simplified map of C's data apace. The DATA AREA includes initialized and uninitialized data, requested memory, free memory. The concept however is eimilar to whet we diecueeed before. The parameters that were passed can now be gotten with the argc and argv. In the actual C program the Stack Pointer is adjusted so that variables can be referenced to it.

Ideally, memory should be aomething that need not be worried about. The Tandy (Microvare) C User's Guide saye, "If not instructed otherwiee, the linker will automatically allocate 1k bytes more the what total size of the program's variables and stringa." If you run IDENT, the OS-9 command that reports information about a module. On DALPHA, you would find that it hae 1355 bytes of memory eet aside. Hardly enough for an array that is 32X150 bytes, plus a handful of integere and stringe. DALPHA requires about 4820 bytes of storage. If you believe the C manual, then there should be an exceas of the required amount. But there isn't! The result is DALPHA goes beyond its allotted area and gete into areas, it shouldn't. The aystem hangs up, a stack overflow occurs and so on.

The linker provides for the amount of memory used by the program, baeed on the needs of linkable modules that go into it. Cstart.r needs 1 byte DP, 80 bytes for data and 896 bytes for atack apace. This is a total of 977 bytes. At link time about 18 modulee are used in DALPHA that come from the C Library, clib.l. Moet do not require any memory, but 6 do. They are chcodes_c(129 bytes), iob_data_c(208), pfldummy_c(3), mem_a(2), and printf_c(36). The total for the library modulee is 378 bytes. Add that to what cstart.r needs and the grand total ie 1355 bytes. And that ia what gets allotted for DALPHA. The main modole gets nothing. If you examine the asaembly listing of it (uee ccl dalpha.c -a), you will see that the paect asks for 0 bytes of atack memory (which it should) and there are no vsects to create data memory.

There is a eolution. The C compiler has an optlon to change the data area eize. The main neede 4820 bytes. Allowing for some breathing room, let's give it 6k more memory. That should be plenty. So when compiling DALPHA try entering:

```
OS9:ccl dalpba.c -m=6k
```

At link time the memory module that is created will have a larger data area, made of the extra memory needed, plus what the other modulea need. When DALPHA runs, you will have enough memory.

One thing should be mentioned for smaller programs, I wouldn't be real worried about the memory situation. Not neceeeearily is all the memory being used at any one time. The cstart.r module has stack memory set aside for all the I/O's and even a "fudge" factor thrown in. But if your programs have many variables, it would be wise to consider memory. The comment in the C manual that the average programmer should not be concerned with memory allocation, should be taken with a grain of salt.

As a peace offering, this month I am giving a new version of DALPHA. This one is similar to the old version since the same routines are used one way or another. There is one important exception. The storage area for the directory is dynamically allocated. There is a system call, sbrk(), which gets free memory from outside the data area. The advantage is any size directory could be sorted. Larger systems with large directories will have more memory available. So this one should work for everyone.

I have also added line numbers to make it easier to transcribe and refer to the program. At David Lynde's suggestion on the lseek() command, I have used uppercase letter "L". The "L" makes the "0" a long integer which is required by the call. Lseek() appears on lines 52 and 162.

The header on the program includes instructions on how to compile. That's on line 12. The "-e=2" tells the compiler to make the module's edition number 2. That way you should be able to tell it from the first in the directory. Running IDENT will tell the modules edition number.

Once again, I am sorry to have caused anyone confusion. I think this second version is more interesting and easier to use. If problems ever do arise with a program, please write me and let me know. Or if you have a question regarding OS-9 drop me a line. I'll see if I can help. Be sure to include a SASE and I'll send you a reply.

In the future, I will talk more about the memory module header and more about C Language memory allocation with the Microware C Compiler. Until then, take care!

```
 1 /* Program to alphabetize directories
 2    By Ron Voigts              December 23, 1985
 3    For 68' Micro Journal, Basic OS-9 Readers
 4
 5    Usage:
 6       OS9:dalpha
 7       OS9:dalpha /d0/A_DIRECTORY
 8       The first example alphabetizes the
 9       current data directory.
10       The second effects a specified directory.
11
12       To compile use: cc1 dalpha.c -e=2
13                                                  */
14
15 #include <stdio.h>
16 #define mask(c)  ((c)&'\137')
17 #define DIR 128
18 #define UPDATE 3
19 #define LENGTH 150
20 #define D_UPDATE DIR+UPDATE
21 #define E_READ 244
22 #define E_MEMFUL 207
23 #define E_WRITE 245
24 #define DSIZE 32
25
26 main(argc, argv)
27 int argc;
28 char *argv[];
29 {
30    int path, count;
31    char *dirname=".";
32    char *q, *sbrk();
33    long p, lseek();
34
35    /* check for too many parameters */
36    if (argc>2){
37       printf("Too Many Parameters\n");
38       exit(1);
39    }
40
41    /* check for alternate directory */
42    if (argc==2)
43       dirname=argv[1];
44
45    /* Open directory */
46    if ((path=open(dirname,D_UPDATE)) == -1) {
47       printf("Can't Open %s\n",dirname);
48       exit(1);
49    }
50
51    /* Directory size */
52    p=lseek(path, 0L, 2);
53
54    /* get some free area for the directory */
55    if((q=sbrk((int)(p))) == -1)
56       exit(E_MEMFUL);
57
58    /* read in directory */
59    count=getdir(path,q);
60
61    /* sort the directory */
62    sortdir(q, count);
63
64    /* write the directory out */
65    putdir(path, q, count);
66
67    /* close the file */
68    close(p);
69
70 } /* end of main program */
71
72 /* reads directory into location
73    pointed to bye entry                */
74 getdir(p, entry)
75 int p;
76 char (*entry)[DSIZE];
77 {
78    int state, c=0;
79    reset(p);
80    while ((state=read(p, entry[c], DSIZE)) > NULL)
81       c++;
82    if (state==-1)
83       exit(E_READ);
84    return(c);
85 }
86
87 /* sorts directory using bubble sort */
88 sortdir(entry, c)
89 char (*entry)[DSIZE];
90 int c;
91 {
92    int i, j, pos;
93    for (i=2; i<c-1; i++) {
94       pos=i;
95       for (j=i+1; j<c; j++)
96          if (compare(entry[pos],entry[j]) > 0)
97             pos=j;
98       if (i != pos)
99          swap(entry[i], entry[pos]);
100   }
101 }
102
103 /* put directory back */
104 putdir(p, entry, count)
105 int p,count;
106 char (*entry)[DSIZE];
107 {
108    int i;
109    reset(p);
110    for (i=0; i<count; i++)
111       if (write(p, entry[i], DSIZE) == -1)
112          exit(E_WRITE);
113 }
114
115
116
117
118 /* compare two directry entries */
119 compare(s, t)
120 char *s, *t;
121 {
122    int i;
123    if (s[0] == '\0')
124       return(1);
125    if (t[0] == '\0')
126       return(-1);
127    i=0;
128    while ( mask(s[i]) == mask(t[i])){
129       if (s[i] > '\177')
130          return(0);
131       else if (t[i] > '\177')
132          return(1);
```

```
133         i++;
134     }
135     return(mask(s[i])-mask(t[i]));
136 }
137
138
139 /* swamp two entries */
140 swap(s, t)
141 char *s, *t;
142 {
143     char temporary[DSIZE];
144     copy(s, temporary);
145     copy(t, s);
146     copy(temporary, t);
147 }
148
149 /* copies s1 into s2 */
150 copy(s1, s2)
151 char *s1, *s2;
152 {
153     int i;
154     for (i=0; i<DSIZE; i++)
155         s2[i]=s1[i];
156 }
157
158 /* return file pointer to start */
159 reset(p)
160 int p;
161 {
162     lseek(p, 0L, 0);
163 }
164
```

"C" User Notes

E. M. (Bud) Pass, Ph.D.
Computer Systems Consultants
1454 Latta Lane, N. W.
Conyers, GA 30207
404-483-1717/4570

INTRODUCTION

This chapter continues the discussion of the design and implementation of a portable text editor by beginning the development of a modal for interfacing terminals and printers. It also provides some information on the newer versions of the Windrush-McCosh C compilers for FLEX. The example C program assists the user in converting TSC word-processor format files to STYLO format.

WINDRUSH-MCCOSH C COMPILER

The latest (as of December 1985) version of the Windrush-McCosh C compilar, number 26.2:1, is available. However, it requires a MEMEND of $BFFF, precluding the loading of any auxiliary programs, drivers, etc., into user memory. This makes its use inconvenient or impossible on many systems. New versions eliminating this problem, allowing split lines and allowing long strings are promised but no release date is available at this time.

** New versions are now released and do not have the $BFFF restriction ** DMW

For those with an interest in such matters, Windrush provided the following update history (restricted to version 26 only):

26.0:0	10/17/84 added unsigned char type declaration
26.0:1	01/11/85 fixed assignment ops and register vars
26.0:2	01/22/85 fixed binary ops with reg var on rhs
26.0:3	01/27/85 optimized chars and zero constants
26.0:4	01/30/85 fixed complex expression evaluation
26.1:0	03/14/85 added _FILE and _LINE macros
26.1:1	03/22/85 fixed while parser
26.1:2	02/05/85 made loader compatible with ar under quix
26.1:3	06/10/85 fixed complex types declaration
26.1:4	08/01/85 fixed FLEX standard library fixed preprocessor error handling fixed error writing temp file
26.1:5	08/25/85 fixed sizeof(expression)
26.1:6	10/02/85 provided correct stdio.h with optimizer
26.2:0	11/10/85 required MEMEND of $BFFF

TERMINAL AND PRINTER INTERFACE MODEL

If an editor is to be capable of being used in a robust manner with a variety of terminal and printer devices, interface models must be developed and implemented. The development and implementation of a model which will support optimized use of most features of most printers and terminals is a difficult task. Unfortunately, the existence of many commercially-available editors which work properly with only a small number of types of terminals and printers demonstrates that such a model was never envisioned during their development.

Features of terminals and printers which should be modelled include:
such elementary items as the following:
- the number of rows and columns,
- how to clear the screen,
- how to sound the alarm,
- how to page-eject,
- if the cursor automatically moves to the next line after reaching the last column,

such non-elementary items such as the following:
- how many pages of memory,
- how to set the cursor position,
- how to read the cursor position,
- how to move the cursor relative to its position,
- how to set screen and cursor attributes,
- how to use alternate fonts,
- how to turn printer port on and off,
- if a terminal can insert and delete characters,
- if a terminal can insert and delete lines,
- if a terminal has local edit modes,

and such problems as the following:
- it is a Beehive or Hazeltine terminal,
- it cannot perform a cr without a lf,
- it has no page-eject,
- it cannot erase screen attributes

without clearing the screen,
it ignores lf after an auto-wrap,
it requires padding characters for time delays,
etc.

Rather than develop still another model, this editor uses the terminfo and termcap methods of describing terminals and printers, as used on UNIX and certain related systems, although with variations. Terminfo and termcap descriptions are lists of capabilities, padding requirements, initialization sequences, and descriptions of valid operations and how to perform them on named devices.

Terminfo files each describe one class of device, in compiled format, and are organized into a tree-structured directory using the first character of the device name as a second-level-node name.

Following is a list of terminfo entries, including the names by which pointers to the entries may be referenced in C programs on these systems:

Pointer Name	Terminfo	Description
Boolean (indicate that the device has some feature):		
auto_left_margin	bw	Cubl wraps from col 0 to last col
auto_right_margin	am	Terminal has automatic margins
beehive_glitch	xsb	Beehive (f1=escape f2=ctrl C)
ceol_standout_glitch	xhp	Standout not erased by overwriting
eat_newline_glitch	xenl	Newline ignored after 80 cols
erase_overstrike	eo	Can erase overstrike with blank
generic_type	gn	Generic line type
hard_copy	hc	Hardcopy terminal
has_meta_key	km	Meta key
has_status_line	hs	Extra status line
insert_null_glitch	in	Insert mode distinguishes nulls
memory_above	da	Display retained above screen
memory_below	db	Display retained below screen
move_insert_mode	mir	Safe to move while in insert mode
move_standout_mode	msgr	Safe to move in standouc modes
over_strike	os	Terminal overstrikes
status_line_esc_ok	eslok	Escape can be used on status line
teleray_glitch	xt	Tabs destructive magic so char
tilde_glitch	hz	Hazeltine; can't print tilde
transparent_underline	ul	Underline character overstrikes
xon_xoff	xon	Terminal uses xon/xoff handshaking
Numbers (provide the value of some device parameter):		
columns	cols	Number of cols in line
init_tabs	it	Tabs initially every # spaces
lines	lines	Number of lines on screen or page
lines_of_memory	lm	Lines of memory
magic_cookie_glitch	xmc	Dead chars left by smso or rmso
padding_baud_rate	pb	Lowest speed with cr/nl padding
virtual_terminal	vt	Virtual terminal number
width_status_line	wsl	Cols in status line
Strings (provide a sequence to perform some operation):		
back_tab	cbt	Back tab
bell	bel	Audible signal (bell)
carriage_return	cr	Carriage return
change_scroll_region	csr	Change to lines #1 through #2
clear_all_tabs	tbc	Clear all tab stops
clear_screen	clear	Clear screen
clr_eol	el	Clear to eod of line
clr_eos	ed	Clear to end of display
column_address	hpa	Set cursor col
command_character	CC	Term settable cmd char in prototype
cursor_address	cup	Relative cursor to row #1 col #2
cursor_down	cud1	Down one line
cursor_home	home	Home cursor (if no cup)
cursor_invisible	civis	Make cursor invisible
cursor_left	cub1	Move cursor left one space
cursor_mem_address	mrcup	Memory relative cursor addressing
cursor_normal	cnorm	Make cursor appear normal
cursor_right	cuf1	Non-destructive space (cursor right)
cursor_to_ll	ll	Last line first col
cursor_up	cuu1	Upline (cursor up)
cursor_visible	cvvis	Make cursor very visible
delete_character	dch1	Delete character
delete_line	dl1	Delete line
dis_status_line	dsl	Disable status line
down_half_line	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode	smacs	Start alternate character set
enter_blink_mode	blink	Turn on blinking
enter_bold_mode	bold	Turn on bold (extra bright) mode
enter_ca_mode	smcup	String to begin programs that use cup
enter_delete_mode	smdc	Delete mode (enter)
enter_dim_mode	dim	Turn on half-bright mode
enter_insert_mode	smir	Insert mode (enter)
enter_protected_mode	prot	Turn on protected mode
enter_reverse_mode	rev	Turn on reverse video mode
enter_secure_mode	invis	Turn on blank mode (chars invisible)
enter_standout_mode	smso	Begin stand out mode
enter_underline_mode	smul	Start underscore mode
erase_chars	ech	Erase #1 characters
exit_alt_charset_mode	rmacs	End alternate character set
exit_attribute_mode	sgr0	Turn off all attributes
exit_ca_mode	rmcup	String to end programs that use cup
exit_delete_mode	rmdc	End delete mode
exit_insert_mode	rmir	End insert mode
exit_standout_mode	rmso	End stand out mode
exit_underline_mode	rmul	End underscore mode
flash_screen	flash	Visible bell
form_feed	ff	Hardcopy terminal page-eject
from_status_line	fsl	Return from status line
init_1string	is1	Terminal initialization string
init_2string	is2	Terminal initialization string
init_3string	is3	Terminal initialization string
init_file	if	Name of file containing is
insert_character	ichl	Insert character
insert_line	il1	Add new blank line
insert_padding	ip	Insert pad after character inserted
key_backspace	kbs	Sent by backspace key
key_catab	ktbc	Sent by clear-all-tabs key
key_clear	kclr	Sent by clear screen or erase key
key_ctab	kctab	Sent by clear-tab key
key_dc	kdchl	Sent y delete character key
key_dl	kdll	Sent by delete line key
key_down	kcudl	Sent by terminal down arrow key
key_eic	krmir	Sent by rmir or smir in insert mode
key_eol	kel	Sent by clear-to-end-of-line key
key_eos	ked	Sent by clear-to-end-of-screen key
key_f0	kf0	Sent by function key f0
:	:	:
key_f10	kf10	Sent by function key f10
key_home	khome	Sent by home key
key_ic	kichl	Sent by ins char/enter ins mode key
key_il	kill	Sent by insert line
key_left	kcubl	Sent by terminal left arrow key
key_npage	knp	Sent by next-page key
key_ppage	kpp	Sent by previous-page key
key_right	kcufl	Sent by terminal right arrow key
key_sf	kind	Sent by scroll-forward/down key
key_sr	kri	Sent by scroll-backward/up key
key_stab	khts	Sent by set-tab key
key_up	kcuul	Sent by terminal up arrow key
keypad_local	rmkx	Exit keypad transmit mode
keypad_xmit	smkx	Enter keypad transmit mode
label_f0	lf0	Label on function key f0 if not f0
:	:	:
label_f10	lf10	Label on function key f10 if not f10
meta_on	smm	Enter meta mode
meta_off	rmm	Exit meta mode
newline	nel	Newline (like cr then lf)
pad_char	pad	Pad character
parm_dch	dch	Delete #1 chars
parm_delete_line	dl	Delete #1 lines
parm_down_cursor	cud	Move cursor down #1 lines
parm_ich	ich	Insert #1 blank chars
parm_index	indn	Scroll forward #1 lines
parm_insert_line	il	Add #1 new blank lines
parm_left_cursor	cub	Move cursor left #1 spaces
parm_right_cursor	cuf	Move cursor right #1 spaces
parm_rindex	rin	Scroll backward #1 lines
parm_up_cursor	cuu	Move cursor up #1 lines
pkey_key	pfkey	Prog funct key #1 to input string #2
pkey_local	pfloc	Prog funct key #1 to exec string #2
pkey_xmit	pfx	Prog funct key #1 to xmit string #2
print_screen	mc0	Print contents of screen
prtr_off	mc4	Turn off printer
prtr_on	mc5	Turn on printer
repeat_char	rep	Repeat char #1 #2 times
reset_1string	rs1	Reset terminal
reset_2string	rs2	Reset terminal
reset_3string	rs3	Reset terminal
reset_file	rf	Name of file containing reset string
restore_cursor	rc	Restore cursor to position of last sc
row_address	vpa	Set absolute row address
save_cursor	sc	Save cursor position
scroll_forward	ind	Scroll text up
scroll_reverse	ri	Scroll text down
set_attributes	sgr	Define video attributes
set_tab	hts	Set tab in all rows current col
set_window	wind	Window lines #1-#2 cols #3-#4
tab	ht	Tab to next 8-space hardware tab stop
to_status_line	tsl	Position cursor to status line col #1
underline_char	uc	Underscore one char and move past
up_half_line	hu	Half-line up (reverse 1/2 linefeed)

in which #n represents parameter n.

Cursor addressing and other strings requiring parameters are designated by parameterized strings with printf-like sequences. For example, to set the cursor, cup is specified assuming parameters row and column, each base zero. The parameter mechanism uses a stack and special codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and eventually output it. These codes have the following interpretations:

```
%%        output `%'
%c %d %s  output pop() as in printf
%p[1-9]   push nth parameter
%P[a-z]   set variable [a-z] to pop()
%g[a-z]   push variable [a-z]
%'c'      push char constant c
%{nn}     push integer constant nn
%+ %- %* %/ %m %& %| %^ %= %> %<
          push(pop() op pop())
%! %~     push(op pop())
%i        add 1 to first two parameters
%? expr %t thenpart %e elsepart %;
          if-then-else (%e elsepart optional)
```

Following is the terminfo description for an adm-3a terminal:

```
la|adm3a|3a|lai adm3a,
 cr=^M, cudl=^J, nl=^J, bel=^G,
 am, cubl=^H, bs, cup=\E=%+ %+ ,
 clear=^Z, cols#80, home=^^,
 lines#24, cufl=^L, cuul=^K,
```

in which the first line provides alternative device names, the equal and pound symbols provide values for numeric and string entries, carets introduce control characters, and the string entry starting with cup= indicates how to set the cursor on an adm-3a terminal (escape, =, (row + ' '), (col + ' ')).

Termcap files are composed of groups of device descriptions, in symbolic format, without the terminfo directory structure.

Entries in termcap follow the same general format as entries in terminfo, although the entry names are restricted to two characters.

Following is the termcap description for an adm-3a terminal:

```
la|adm3a|3a|lai adm3a:\
 :cr=^M:do=^J:nl=^J:bl=^G:\
 :am:le=^H:bs:cm=\E=%+ %+ :\
 :cl=^Z:co#80:ho=^^:li#24:\
 :ma=^K^P:nd=^L:up=^K:
```

The terminfo file format is newer than the termcap file format and is capable of more efficient use, since it does not require searching, scanning, and processing on each use, as does the termcap format.

Operational models based on the terminfo and termcap device descriptions are presented in the next chapter.

C PROBLEM

The C functions in the test program below compute the factorial function in both recursive and non-recursive manners. Note that float or double types could have been used (although carefully) in place of long, in this case, despite the requirement that the results be calculated exactly. The maximum value calculated correctly may be obtained from the output in comparison with a published table.

In general, the programmer does not have the luxury of precalculated correct results, and should use float or double very carefully (or avoid their use entirely) when dealing with values which must be exact, such as in monetary calculations. For example, use longs to contain exact dollar amounts to the penny, then multiply or divide by 100 when inputting and outputting values for human use, rather than carry the floating-point values internally. As an added bonus, the calculations will normally be performed much more rapidly in integer mode than in real mode.

```
#include <stdio.h>

long recfact(n)
int n;
{
    return((n < 2L) ? 1L : (recfact(n - 1) * n));
}

long nrecfact(n)
int n;
{
    long f = 1L;

    while (n > 0)
    {
        f = f * n;
        --n;
    }
    return f;
}

main()
{
    int i;

    for (i = 0; i < 20; ++i)
        printf("%3d     %13ld        %13ldn",
            i,recfact(i),nrecfact(i));
    pflinit(); /* required for mccosh c only */
}
```

For the next C problem, write a C program which will exactly generate and output the elements all of the elements of the Fibonacci sequence, with any given number of digits of precision. The Fibonacci sequence is defined for purposes here as follows:

the first element has value zero,
the second element has value one,
each element after the first two has
    the value of the sum of the preceding
    two elements.

Thus, the first few elements are 0, 1, 1, 2, 3, 5, 8, 13. Stop the calculation on the first value exceeding the number of digits of precision specified. Do not assume that this number of digits of precision is within the range of longs or doubles.

EXAMPLE C PROGRAM

Following is this month's example C program; it converts TSC word-processor-formatted files into STYLO format, although it does not convert all of the TSC commands nor any of the TSC registers. This conversion is made necessary because STYLO provides no direct means of loading files into its format.

```
/*  stylfilt.c

    converts tsc word processor format to stylo format

     tsc wp       stylo
    --------     -------
    cr cr        cr cr
    cr .         cr ,
    cr           space if not in .nf or .ce n
    cr           cr if in .nf

    .ce n        ,ce n
    .fi          ,ju
```

```
    .in n       ,in n
    .nf         ,nj
    .pg         ,PG
    .ai n       ,ai n
    .ap n       (n + 1) cr
    .xx ...     ,xx ...  (may be incorrect atylo command)

    \\          \
    \#          #
    \@          @

*/

#include <stdio.h>
#include <ctype.h>

main(argc,argv)
int argc;
char **argv;
{
    FILE *input = stdin, *output = stdout;
    char *p, string[256], string1[256];
    int c = 1, d = 0, l = 0, pc = '\n';
    int ce = 0, nf = 0, sp = 0, zz = 0;

    putc('\n', stderr);
    if (argc > 1)
    {
        if (!(input = fopen(*++argv, "r")))
        {
            fputs ("can't open input\n", stderr);
            exit (1);
        }
    }
    if (argc > 2)
    {
        if (!(output = fopen(*++argv, "w")))
        {
            fputs ("can't open output\n", stderr);
            exit (1);
        }
    }
    while ((c = getc(input)) != EOF)
    {
        switch (c)
        {
        case '\\':
            pc = '\\';
            if ((c = getc(input)) == EOF)
                break;
            if (c == '\n')
                putc('\\', output);
            else
            {
                putc(l = c, output);
                break;
            }
        case '\n':
            if (l = ((pc == '\n') || nf || ce))
            {
                putc(l = '\n', output);
                if (ce)
                    --ce;
            }
            else
            {
                pc = c;
                if (((c = getc(input)) == EOF) ||
                    (c == '\n') || (c == '.'))
                    putc(l = '\n', output);
                else
                    putc(l = ' ', output);
                if (c != EOF)
                {
                    ungetc(c, input);
                    c = pc;
                }
            }
            break;
case '.':
    if (pc != '\n')
    {
        putc(l = c, output);
        break;
    }
    if (fgets(string, 256, input))
    {
        for (pc = zz = sp = ce = 0, p = string;
            (c = *p); ++p)
        {
            d = tolower(*(p + 1));
            switch (c)
            {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                zz += (zz * 10) + (c - '0');
                break;
            case 'C':
            case 'c':
                if ((d == 'e') && (!pc))
                {
                    ++pc;
                    ++ce;
                }
                break;
            case 'F':
            case 'f':
                if ((d == 'i') && (!pc))
                {
                    ++pc;
                    nf = 0;
                    *p += ('j' - 'f');
                    *(p + 1) += ('u' - 'i');
                }
                break;
            case 'N':
            case 'n':
                if ((d == 'f') && (!pc))
                {
                    ++pc;
                    nf = *(p + 1) += ('j' - 'f');
                }
                break;
            case 'S':
            case 's':
                if ((d == 'p') && (!pc))
                {
                    ++pc;
                    ++sp;
                }
                break;
            }
        }
        if (!zz)
            zz = 1;
        if (ce)
            ce = ++zz;
        if (sp)
            while (zz--)
                putc('\n', output);
        else
        {
            putc('.', output);
            fputs(string, output);
        }
        pc = l = '\n';
    }
    c = '\n';
    break;
```

```
        default:
            putc(l = c, output);
        }
        pc = c;
    }
    if (input != stdin)
        fcloae(input);
    if (output != stdout)
        fcloae(output);
    exit (0);
}
```

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

A Top-Level Menu

I use my computer for many things and I have a rats' nest of directories on my disk to prove it. Some of the things I work on most often are dowo six or more levels in the directory tree. I get tired of typing commands like:

OS9: chd /h0/source/tools/txt/doc just to get to the directory where the documentation I'm working on is stored.

Last summer I spent most of my time working on one big software project. I moved around in one small branch of my directory structure. After a few weeks I got tired of typing a long CHD command to get to that branch every time I logged on. I realized that I could put the directory into the Password file as my logon directory. It worked fine, saving me some time and lots of annoyance over the summer.

During the school year my activities are not so limited. I log onto the department's computers; I write these columns; I work on things that may turn into books; I develop software; I do accounting, write letters, and never seem to use the same directories two days in a row. On top of that I have a wife who sometimes shares my computer. She certainly doesn't know where all the treasure is buried.

Since moving to the correct directory and invoking the right program is repetitive detail work, it is a suitable task for a computer (not a human). I wrote a program. I call it Menu because it lets me present the user (me usually) with a menu that allows him to make selections limited only by my creativity when I write the menu control file. Menu will stay around under the user like a personal aysgo module.

The menu program is simple. Menu doesn't know anything about formatting screens except that a carriage return moves the cursor to a new line. It doesn't know anything about OS-9 commands except that the "system" function invokes them.

I got tired of typing a long CHD command to get to that branch every time I logged on.

I realized that I could put the directory into the password file as my logon directory.... It worked fine....

Menu is controlled by a file. The first part of the control file is displayed on the screen. I put the menu, a prompt, and lots of screen formatting codes in that part of my control files. The display part of the menu is terminated by a line containing just three dollars signs, $$$. The second part of the control file contains commands. Each line starts with a single character selector which is followed by a shell command. If the user types the selector for a line, Menu will run the shell command then go back and display the menu again.

It is important that Menu redisplays the menu and waits for another selection after a command completes. It means that Menu stays around to help you when you finish your task. (It also means that Menu ties up some memory.)

One selector character is special. If there is a command line for the character "q", Menu will exit when it completes that command.

The command lines in the control file are limited to eighty characters each. This is a limitation of the system function, but it isn't an important problem. If you want a menu selection to do something too complicated for a short command line, invoke shell on a file with a shell script in it. If you really want to get fancy, menu will work as a hierarchical menu system. Each selection that points to another menu gets a command line like:

KMP Knuth Morris Pratt algorithm really shines....

Boyer Moore algorithm gets a little carried away....

A simpler algorithm that works almost as well as Boyer Moore and has less overhead is the mismatched character algorithm....

menu Special.Menu.c7 When you quit from a menu you pop back to the menu that called it.

My current menu files are so full of ANSI control sequences for changing attributes and drawing boxes that they don't make easy reading, but here's any early example that reads pretty well.

```
^[[2J

          ^[[4mM a i n   M e n u^[[0m

     ^[[45;8 p
     ^[[7mt^[[0m          Enter text
     ^[[7ml^[[0m          Write letters
     ^[[7mp^[[0m          Lookup phone numbers and
addresses
     ^[[7mr^[[0m          Logon to a remote system
     ^[[7ms^[[0m          Simple logon (nothing special)
     ^[[7mq^[[0m          Quit
```

Make a selection: $$$ tchd /h0/CATH/TEXT; ds ; * start dynastar lchd /h0/CATH/LETTERS; ds ; * start dynastar pchd /h0/PEOPLE ; * don't start anything but a shell rdt ; * run my terminal program schd /h0; qecho; echo Logging off

Notice that the selections are t, l, p, r, s, and q. Any other selection will draw an error message and another chance. The length of time for which the error message will be displayed is determined by the SLEEPTICKS constant in the program. It is set to show for a nice interval on my Level Two system. On a Level One system (with its slower tick rate) you should make that constant smaller, or you'll see the error message until you are very sick of it.

A good place to run the Menu program from is the Password (or startup) file. In the password file, use Menu as the initial program.

I have been surprised by Menu. It does a lot to make OS-9 easier to use. I expected it to be a shorthand way for me to get around when I logon. I think it may actually prove to be an easy "friendly front end" for OS-9.

There are some changes you might want to make to Menu. The main ones are to set things up so Menu will ignore interrupts and EOF. I didn't do that because I fool with the program a lot and want to be able to kill it easily.

Fast C Functions
Continued

A few months ago I presented assembly language versions of the basic functions in Microware C's string handling library. I argued that the most effective use of assembly language was in libraries and operating systems. I left findstr and findnstr out of the set of functions because they are hard to do right. Microware seems to agree with me; they note in the C manual that the algorithm they used is not the most efficient. The standard (not easy) string matching algorithms are called the Knuth Morris Pratt Algorithm and the Boyer Moore Algorithm. An even flashier algorithm was invented by one of my professors, Joel Seiferas. The problem with the fancy algorithms is that they have a high overhead. For simple jobs they aren't worth using.

Lets say we're looking for a pattern in a string called data. The simple way to match strings is to take the pattern and compare it against substrings of data until you find a match or reach the end of data. I'd guess that Microware C's library findstr function works that way. If the pattern is short, unrepetitive, or unlike the data string this is not a bad way to do it. We run into trouble when these conditions aren't met.

Take the following (exaggerated) example: The pattern is "abababsbb" and the data string is "abababab...ababbabab". You can see that the pattern will match the data string at the fifth to last position it will try. Each time it compares the pattern to a substring of the data string the simple method will compare "abababab" to "abababab" before it finds "b" not equal to "a".

The trick in the Knuth Morris Pratt (KMP) algorithm is that in this example it would note that if the final "b" meets an "a" we reject at the position we were checking, move over two and restart the matching operation looking for the second to last "b" in the pattern. The "abababa" that we've already looked at doesn't have to be checked again. Since this algorithm never has to back up and check a character from the data string again it is particularly nice for doing things like searching a file for a long string.

The KMP algorithm really shines when the pattern is long and repetitive. The repetition in the pattern doesn't have to be complicated. A pattern with a hundred blanks at the beginning is a fine example of a repetitive pattern. If the beginning of the pattern doesn't match often, the algorithm doesn't get a chance to use its power often. Since most searches aren't for long repetitive strings KMP searching isn't generally the best way to search. Still, we can't write KMP off that easily. If we get fancy and permit things like wild cards in the pattern, KMP-type algorithms need to be looked at again.

The Boyer Moore Algorithm is based on the idea that looking at the pattern backwards will often let you jump through the data string quickly. If the pattern was "abb" and the data was "abcbabababba" comparing the third byte of the pattern with the third byte of the data would show that the pattern doesn't match at position one. In fact, since "c" doesn't appear anywhere in the pattern, we can rule out matches at positions one, two, and three.

The Boyer Moore algorithm gets a little carried away when it is preprocessing the pattern. A simpler algorithm that works almost as well as Boyer Moore but has less overhead is the mismatched character algorithm. I have included C code for it with this column. I would have given you assembly language instead, but the algorithm is hardly easy to see in C. In assembler it

would be almost impenetrable (also long). Try converting it to assembly language yourself or petition Don W. for it. I have suggested to him that my assembly language functions might make a good disk for him to sell.

The mismatched character algorithm starts comparing at the right side of the pattern and the spot in the data string where the pattern would end. It compares toward the left until it reaches the beginning of the pattern (Match!) or a mismatch. When it gets to a mismatch it looks at the character after where it started in the data string. It chooses a new starting point in the data string which would let that character fit into the pattern. (If it could fit in several spots it picks the first spot. If it doesn't appear in pattern it jumps by the length of the pattern plus one.) The algorithm loops until it finds a match or runs out of data string.

If the data string is short the simple algorithm is fastest because of the setup time for the other methods. The mismatched character algorithm is good when the data string has lots of different characters in it and the pattern is more than two or three characters long.

- - -

```
 1 #include <stdio.h>
 2 #include <sgstat.h>
 3
 4 #define TRUE 1
 5 #define FALSE 0
 6 #define LINESIZE 133
 7 #define SLEEPTICKS 100
 8 #define ENDFLAG "$$$\n"
 9 #define ESCAPE_CODE 'q'
10
11 static direct char *MenuFName="/H0/Startup.Menu";
12 static direct char c;
13 static direct FILE *MFile;
14 static struct sgbuf Options, OldOptions;
15
16 main(argc, argv)
17     int argc;
18     char **argv;
19     {
20         char line[LINESIZE];
21         register char *ptr;
22
23         setbuf(stdin, NULL);
24
25         if(argc > 1)
26             MenuFName = argv[1];
27         getstat(0, 0, &Options);
28         _strass(&OldOptions, &Options, sizeof Options);
29         Options.sg_pause = 0;
30         /* Here we could mess with
31             sg_eofch, sg_pzch, sg_kbrch, and sg_kbach
32             if we wanted to prevent anyone from slipping
33             out of the menu by mistake
34         */
35         while(TRUE){
36             if((MFile = fopen(MenuFName, "r")) == NULL){
37                 fprintf(stderr, "Menu file %s won't open.  Error %d\n", er
rno);
38                 exit(0);
39             }
40             while(TRUE){
41                 setstat(0, 0, &Options);
42                 while(fgets(line, LINESIZE, MFile) != NULL){
43                     if(strcmp(line, ENDFLAG) == 0)
44                         break;
45                     /* clear the CR off the line */
46                     for(ptr = line; *ptr && (*ptr != '\n'); ++ptr);
47                     *ptr = '\0';
48                     printf("\n%s", line);
49                 }
50                 c = getchar();
51                 while(fgets(line, LINESIZE, MFile) != NULL)
52                     if(*line == c){
53                         fclose(MFile);
54                         setstat(0, 0, &OldOptions);
55                         system(line+1);
56                         break;
57                     }
58                 if(c == ESCAPE_CODE){ /* escape */
59                     exit(0);
60                 }
61                 if(c != *line){
62                     printf("\nOption %c not defined.  Try Again\n", c);
63                     tsleep(SLEEPTICKS);
64                     rewind(MFile);
65                     continue; /* redisplay */
66                 }
67                 break; /* back to reopen the menu file */
68             } /* end while TRUE */
69         } /* end outer while TRUE */
70         exit(0);
71     }
```

```
 1 #include <stdio.h>
 2
 3 fsinitskip(pat, skip)
 4   char *pat;
 5   int skip[256];
 6   {
 7       int i,j;
 8       register int *iptr;
 9
10       i = strlen(pat);
11       for(j=0, iptr = skip; j<256;++j)
12           *iptr++ = i;
13       for(i--;*pat;++pat, i--)
14           skip[*pat] = i;
15       return;
16   }
17
18 findstr(pos, str, pat)
19   int pos;
20   char *str, *pat;
21   {
22       int length, slength, last;
23       int strptr,patptr;
24       int skip[256];
25
26       fsinitskip(pat, skip);  /* initialize the skip array */
27       str = str + pos - 1;  /* Skip to offset pos in str */
28       length = strlen(pat);
29       last = length - 1;
30       slength = strlen(str);
31       strptr = patptr  = length - 1;
32
33       do{
34           if(str[strptr] == pat[patptr]){
35               /* matched a character */
36               strptr--;
37               patptr--;
38           }else{
39               /* mismatch */
40               strptr += length - patptr + 1; /*point strptr one to the
41                                                 right of where it start for
42                                                 this pass */
43               patptr = last;  /* Hop back to the right end of pat */
44
45               if(skip[str[strptr]] > length - patptr + 1)
46                   strptr += skip[str[strptr]] - (length-patptr+1);
47           }
48       } while((patptr>=0) && (strptr<slength));
49       if(patptr < 0)
50           return strptr+2;
51       else
52           return 0;
53   }
```

QPL

Jim Loe, CPU - Compiler Products Unlimited, Inc.

QPL - A Bold Step to Very High Level Language

This is the first in a series of articles about a Very High Level Language named QPL. It is available from Compiler Products Unlimited for systems running FLEX.

Because QPL is a very-high-level language, it is very different from most popular languages in use on FLEX systems. Other very-high-level languages which you may have heard of are Lisp, Snobol4, Easytrieve, and Prolog. Learning a new language involves some effort; however the payoff is the ability to produce completed programs much faster than in other languages.

Two basic differences between QPL and Pascal-type languages are convenience and power. Convenience is a relative measure of how easy it is to do something in a language. For example, not being able to address an array with a real number is inconvenient.

Language power is being able to do a lot with few statements. For example, being able to copy array DAYS to array APPOINTMENTS by an assignment statement APPOINTMENTS = DAYS is more powerful than having to write a loop to do it.

The effect of having a powerful, convenient language is three fold; programs get written much faster, and the user perceived quality of the programs is higher, because better tools produce better results. While the tools / result-quality is difficult to quantify, an example can illustrate it; paint two car fenders, the first one with a Q-tip and a can of paint, and second one with a spray gun. The third effect is that a more powerful tool gives you the confidence to do more complex programs.

Power and convenience are obtained in QPL by generality and simplicity. An example of generality is that in QPL, an array may contain a mixture of basic data types, and it may be addressed by real numbers (real numbers are the only kind in QPL). Generality promotes solutions which have the same 'shape' as the problem. For example, write a program which makes a list of words used in text file, and records the number of times each is used. In QPL the program can contain a two-dimension array which has the words found in the file, and the number of times each word was found. To do this in other languages would require two arrays, one for the words, and another for the number of times used.

Because QPL arrays can contain mixed data types, they can do the job of Pascal records. In fact, since a QPL array element can contain any type of data of any size, it can do a bigger job than records. This demonstrates how generality in QPL features allows simplicity in the language.

QPL DESCRIBED:

QPL consists of two major parts; a relatively conventional group of abilities for doing arithmetic, making comparisons, and conditional branching and looping. The second major part is a group of abilities for creating patterns and doing pattern matching. The conventional abilities are provided in slightly unconventional ways, which produces some of the power advantage. The real power in the language comes in the pattern-processing functions.

The first group of abilities is compatible with 'linear' thought processes and is the most conventional part of QPL. We will discuss this group of features first.

INPUT and OUTPUT:

QPL has three pre-defined variables: INPUT, OUTPUT, and NULL. The variable OUTPUT is used to cause something to be printed on the console. Thus the statement:

```
OUTPUT = "Hello world"
```

will print 'Hello world' on your terminal. The statement

```
OUTPUT = 123.5 + 3.1
```

will print 126.6 on your terminal. Thus we see that anything assigned to OUTPUT will be printed out. OUTPUT can also be used like any other variable. For instance,

```
OUTPUT = OUTPUT + 1
```

will print 127.6 if executed after the previous statement.

The variable INPUT is a signal to the compiler to get a string from the terminal keyboard. The statement

```
SENTENCE = INPUT
```

will cause the compiler to get a line of text from the keyboard (terminated by carriage return).

There is no limit to the size of string which can be assigned to the variable SENTENCE (or to any other variable), and there is no limit to the size of string that INPUT can get. This 'no limits' approach is central to QPL and is one of the reasons for high productivity. There is seldom a need to concoct program tricks to bypass limits imposed by the language.

There is one limit no language can escape, the size of available memory, and this limits the size of all data items.

NULL is just a zero-length string, which is often a handy initial value to assign to variables. It is also the default value of all variables. NULL can be used in string operations, but it cannot be used in number operations (produces error message).

USER VARIABLES:

User variables require no explicit declaration, except for arrays which must be named and sized. Unlike other languages which requires arrays to be declared with fixed sizes, QPL arrays may have variable size, determined at run time. All variables, including array elements, can hold any value such as a number, string, or a pattern, of any size.

NUMBERS:

Numbers are strings which have only numeric characters, including decimal point. Numbers are accepted in INPUT statements, or hard coded in the

program, in either of two formats: decimal and exponential. Example decimal numbers are 12, 99.67, 12345678000320303012021 30.0272387. Note that numbers may have a large number of significant digits. All digits are processed by the arithmetic routines, producing EXACT RESULTS. There is no rounding or truncation. Exponential numbers consist of a number followed by 'E', followed by the power of 10 exponent. An example exponential number is 123E3450. Numbers have a range from 10 exponent 32000 to 10 exponent -32000. In exponential format numbers, only the digits to the left of the 'E' are significant in arithmetic functions. This fact allows programmer control of number precision independent of number value. Two numbers with the same value but different precision are 50E4 and 500000. The arithmetic functions process 50E4 much more quickly than 500000, and produce results with lower precision.

ARITHMETIC:

QPL arithmetic looks fairly conventional, as the calculation of the weight of an aluminum atom in the example below shows:

```
ALUMINUM = 26.97 / 6.023E23
```

Note that there is a space on either side of the division operator '/'. The space is not just for clarity, it is required for all two-operand operators. The two-operand arithmetic operators are /, *, +, -, and they have the conventional meanings. The only single-operand arithmetic operator is negation, and it must not have a space between it and its operand. An example of negation is as follows:

```
SAM = -SAM
```

Numbers can be output (printed) in two formats, decimal and exponential. Selection of the format is done by setting the EXPO flag through a call to the function EXPO.

```
EXPO(1)  -- sets for exponential output
EXPO(0)  -- sets for decimal output
```

NUMBER PRECISION:

Because the arithmetic system produces exact results, it is possible to create numbers which are unwieldy. An example of this is as follows:

```
SAM = 1E500 + 1
```

This assigns to SAM the number 1000000(499 zeros)00000001. Even if we set the EXPO(1) switch, the number will be printed with all 499 zeros, and the final 1, because this is the only way to express it. In other words, EXPO(1) does not help for numbers which have a large number of signigficant digits. We can reduce the precision of a number by using the APPROX function:

```
APPROX(SAM,2)
```

This will truncate the precision of SAM to about 2 digits, allowing it to be printed as 100E498, if EXPO(1) has been set.

Thus QPL's arithmetic system can produce exact results without rounding errors for business applications. It can also produce very high-precision results for scientific work, with truncation explicitly controlled by the programmer.

COMPARISON:

Comparison in QPL is done by comparison functions with names like LE, GT, LT. There are 8 total (2 lexical) and are used as follows:

```
IS = LT(COUNT,3)
```

In the statement above, the function LT compares the value of COUNT with 3. If COUNT is less than 3, the function LT returns a value of 1, which gets assigned to IS. Otherwise, it returns 0. LT also sets a global success / fail flag which can be used to cause branching. While these comparison functions may seem similar to the comparison operators .LE., .LT., etc of Fortran, the difference is that in Fortran .LE. is an operator, and does not return a value, but sets an internal pass / fail flag, as part of an IF statement.

BRANCHING:

To do branching and looping, QPL uses conditional and unconditional GOTO statements which come in three types:

```
:S(HERE)  -- this is a success goto to 'HERE'
:F(THERE) -- this is a fail goto to 'THERE'
:(YONDER) -- unconditional goto to 'YONDER'
```

In addition, there are two combined forms which look like this

```
:S(HERE)F(THERE)   -- if success goto HERE, else goto
                      THERE
:F(THERE)S(HERE)
```

LABELS:

Labels are the places to which goto statements transfer control. They must begin in the first column and must begin with A-Z.

COMPARISON AND BRANCH:

A complete comparison and branch statement looks like this

```
LT(COUNT,3)   :S(THERE)
```

If the value of COUNT is less than 3, the program will jump (goto) the label 'THERE'. Note that we did not use the returned value from LT in the above example.

The above example can be modified by the inclusion of the name indirection operator '$', so that execution will goto whatever THERE contains. In this case, the code would look like

```
LT(COUNT,3)   :S($THERE)
```

A use for this line of code would be to have a program read in a label to goto from a text file. This technique can save significant amounts of code which would be required as 'case' statements in other languages. If the goto label does not exist, the program aborts with an error message, it does not jump into the 'garbage can'.

The fact that comparison functions return 1 or 0 can be used to perform complex logic in little code. For example, we can test to see if P is not 1, 3, 5, 7, or 8 by the following code:

```
NO = NE(P,1) + NE(P,3) + NE(P,5) + NE(P,7) + NE(P,8)
```

If P satisfys all 5 conditions, the value of NO will be 5.

INDIRECTION:

The name-indirection operator is the dollar sign '$'. It is written without any space between it and its operand. The function of indirection is to get the value of its operand. For example, in the program below

```
SAM = 123
BOB = "SAM"
OUTPUT = BOB
OUTPUT = $BOB
```

line 3 will print SAM, because QPL dereferences the right side by one level just as almost all languages do. Line 4 will print 123, because the indirection operator forces an extra level of dereferencing (dereferencing = value replacement). Indirection is useful for chaining together related data. There is no limit on the level of indirection, except that it must be valid, or an error message is produced.

ARRAYS:

QPL arrays are different from arrays in most languages. Besides the 'hold anything' ability, QPL arrays are created at run time and their size can be determined at run time. You can have a QPL program read FLEX Memend, and size arrays based on space available. A further advantage of QPL arrays has to do with the fact that like all data items in QPL, arrays are

allocated apace aa needed. What thia means is that a large array can be declared without taking up a lot of memory apace. Only when data ia written into the array will apace be allocated. QPL arraya do another apace aaving trick when array elementa are aaaigned the aame value. The trick ia to only inatall one copy of the value in memory, and aet pointera to that value in duplicate entrya.

Aa mentioned earlier, QPL arraya can do the job of named recorda. In named-record access, field namea are used to help program readability. We can acceaa QPL array elementa by name alao, by defining namea having numeric values. An example ia ahown below:

```
ARRAY(CALENDAR,12,31)
JAN = 0
FEB = 1
MAR = 2
CALENDAR<FEB,12> = "Lincoln's birthday"
```

STRING CONCATENATION:

The concatenation operator ia the ampersand '&'. It ia a two-operand operator, ao it muat have a apace on either aide. To concatenate a atring and a number we would write

```
RESULT = NAME & AGE
```

Aaauming that AGE waa a number, it will be converted to a atring and appended to NAME, and the reaulting name and age atring aaaigned to RESULT.

Numbers can alao be concatenated to form new numbers. For example

```
DOZEN = 12
OUTPUT = (DOZEN & DOZEN) + 5
```

will print 1217.

Concatenation ia alao uaed in forming very powerful patterna, and thia will be diacuaaed in the next inatallment.

This firat article gives you a brief aummary of the firat 5 chapters of the QPL language manual. It covers those QPL features which are aimilar to other high-level languagea. Even in these basic features, QPL providea extra power, in waya which are eaay to apply.

The next article will cover pattern proceaaing, which conaiata of pattern generation by uae of alternation and concatenation operators, and the uae of apecial patterna. These features are aimilar to pattern matching functiona found in the Snobol4 language; however they have added power and are much easier to uae.

A High Performance System Using the MC68020

by

Dr. David McCartney
Paul F. Groepler

MOTOROLA INC.

6501 William Cannon Dr. W.
Austin, Tx. 78735-8598

Introduction

A High performance processor system demands a very high performance central processor as part of its design. The new MC68020 32-bit microprocessor from Motorola is such a device. This article will discuss the Main festures of this new advanced microprocessor and how a system is designed using it. The interface to memory and peripherals is described along with real-time support via interrupts. The concepts discussed are then integrated to produce a design for a typical minimum system consisting of central processor, memory and some basic I/O.

MC68020 Features

The MC68020 is the latest member of the M68000 family of processors and peripherals. The design of the basic 68000 processor which was introduced in 1979 has been greatly enhanced and extended to produce the new MC68020, which is the first true 32-bit microprocessor on the market. The new MC68020 32-bit processor is object code compatible with all of the earlier processors in the M68000 family and contains many new additional features which greatly enhance the overall performance of the processor. The features available to users of the new MC68020 processor consist of the following: -

a) Virtual Memory / Machine SupPort
b) Sixteen 32-bit General Purpose Data and Address Registers
c) Two 32-bit Supervisor Stack Pointers
d) 32-bit Program Counter
e) Five Special Purpose Control Registers
f) 4 Gigabyte linear direct addressing range
g) 18 Basic Addressing Modes
h) Memory Mapped I/O
i) Coprocessor Interface
j) High Performance On-Chip Instruction Cache
k) Operations on Seven different data types
l) Complete support for a General Purpose Coprocessor Interface

As shown in Fig 1, the user level programming model for the MC68020 is identical to that of the other M68000 family processors and consists of 16 32-bit general purpose registers, a 32-bit program counter, a 16-bit status register, and a 32-bit user stack pointer register. Fig 2 shows the additional features available to programs running in the supervisor level. This includes a 32-bit Vector Base Register, which allows the processor to have its 1K byte vector table relocated anywhere within the 4 Gigabyte linear address space. Two alternate function registers to allow supervisor code to access any data space. Two 32-bit registers to control the operation of an on-chip cache. Finally, two separate 32-bit supervisor stack pointers to allow the separation of interrupt and task related exception conditions.

Interfacing to memory devices and peripherals is a simple matter with

the MC68020 because the processor is designed to use a technique known as Dynamic Bus Sizing. This concept allows the MC68020 to use either an 8 bit, 16 bit or 32 bit data bus to interface to memory and peripheral devices. The processor can dynamically change the size of its interface data bus on every bus cycle to handle the different sized interface ports. The interface circuitry designed into the MC68020 to cope with the dynamic bus sizing also allows the processor to read operands from any byte boundary in the addressing range of the processor. This feature, known as Operand Misalignment, means that the MC68020 processor will automatically generate the appropriate number of bus cycles to perform an operand transfer. The number of bus cycles generated depends on how the operand is aligned in memory and also on the size of the interface port's data bus.

One of the new features on the MC68020 which increases the performance compared to the MC68000 is the addition of an on-chip cache memory. This is a block of very fast memory which is implemented on the same silicon as the processor and is used to temporarily store all the instruction words prefetched by the processor. This means that when the processor executes a looping branch instruction, it will probably then find that all of the subsequent instructions are stored in the on-chip cache. The MC68020 can therfore execute directly from the cache at a faster rate than if it had to refetch all the instruction words from external memory.
All of the above features and concepts have been described in great detail by MacGregor et al (1) and therfore this article will not attempt to re-iterate this information. Instead it will concentrate on the application of the MC68020 in a system configuration.

Design Considerations

As already discussed, the MC68020 is a very high performance microprocessor which has 4 to 5 times the power of an MC68000. To achieve the maximum performance from the MC68020, care must be used when designing a complete system, so that all bus cycles are performed in minimum time. The MC68020 is designed to perform a minimum 3 clock bus read or write cycle and therfore the memory acccess time should be considered in the attempt to design the system for zero wait state operation. Considering the timing figures given in the MC68020 User's Manual (2) it can be seen that for worst case using a 16.67 Mhz clock signal, there is approx 90ns from when the address strobe is asserted by the processor to when data will be latched internally by the processor. Allowing approximately 20 ns for address decode time, there are 70 ns left for memory access time. This implies that a zero wait state system running at 16.67 Mhz can be designed with memory devices having an access time of 70 ns of better. Memory devices with slightly longer access times could also be used by employing faster logic in the address decode system and by making use of the External Cycle Start (ECS*) signal available on the MC68020. The The ECS* signal appears during State 0 of every bus cycle and therfore can be used to initiate the address decode logic before the address strobe is asserted by the processor. This allows for slightly slower memories to be used and yet still achieve the target of zero wait-state operation.

The diagram in Fig 3 shows how the performance of the M68000 family processors is related to the memory access time. The highest performance is achieved when any of the processors are designed to run with zero wait-states. As the memory access time is increased to the point where wait-states are introduced into the system, the overall performance of the processor drops as a stepwise function. The MC68020 as shown in the diagram has the highest performance when operated at 16.67 Mhz with zero wait-states. When one wait-state is incorporated into the system, there is a dramatic fall-off in performance of the MC68020. This drop in performance is equivelent to losing the power of an 8 Mhz MC68000 from the total system design.

Since the number of bus cycles generated by the processor for each operand transfer is related to the size of the interface port and the way that the data is aligned in memory, these factors influence the performance of the processor. Hence the diagram of Fig 3 for the MC68020 shows a difference between the best case timings and the average worst case timings. The highest performance is achieved by the MC68020 running code at 16.67 Mhz and having all the data values properly aligned to avoid the need for multiple bus cycles.

Memory Interface

The previous section mentioned how the MC68020 32-bit microprocessor is capable of interfacing to memory or peripheral devices via either an 8, 16 or 32-bit data bus. The dynamic bus sizing capabilities designed into the microprocessor will automatically generate the required number of bus cycles needed to complete the operand transfer. Thus the highest performance from an MC68020 system is achieved when the system is designed to use a 32-bit interface bus. This is because internally the MC68020 microprocessor always assumes that it is connected via a 32-bit data bus to the system memory and the processor begins an operand transfer by attempting to transfer 32-bits of information. During the first bus cycle of the operand transfer, the processor indicates that it is attempting a 32-bit transfer by setting the appropriate code on the SIZ0 and SIZ1 output pins. Depending which handshake code is given, the processor will either move on to the next operand transfer if it received a 32-bit handshake or generate generate extra bus cycles in order to complete the operand tranafer if an 8 or 16 bit handshake was received. If a 16-bit interface is used in a system design the processor will generate twice as many bys cycles to transfer a block of 32-bit operands and if an 8-bit bus is used, there will be four times the number of bus cycles generated.

In a system design most activity will occur in the RAM area of memory. Therfore, it makes sense to design the RAM section of the memory to use a 32-bit interface bus. The processor can then read and write up to 32-bits of information during each bus cycle which leads to a very high-performance system. Most large scale systems use the ROM section of the system memory to contain either a low level monitor program or a small bootstrap loader program which is capable of going out to some form of mass storage device and loading a more complex program such as an operating system into the system RAM. Since the ROM code is usually not required very often by the processor, the system designer must decide what size of interface bus to implement. Most of the commonly used available ROMs and EPROMs are organized as 8-bit wide data bus. For more complex systems or higher performance, a 16-bit bus may be used with the slight disadvantage to the system programmer that all the code which will eventually be put in the ROM or EPROM must be 'split' into separate sections containing odd bytes and even bytes. The highest performance is achieved using a 32-bit interface bus where the code must be 'split' into four separate sections to allows four ROMs or EPROMs to be produced.

Dynamic Bus Sizing

The introduction of dynamic bus sizing opens up a choice to the MC68020 system designer. Now, when interfacing 8 and 16-bit peripherals, the designer may choose between a 'pseudo' 32-bit vs. a 'true' 8 or 16-bit interface prot as shown in Fig 4. There are advantages and disadvantages to both methods.

One choice, interfacing to an 8-bit peripheral with a true 8-bit port allows the programmer to have a software interface to the peripheral via 32-bit wide read and write instructions. The processor and the external hardware automatically handle the task of dynamically sizing the bus to transfer 32 bits of data across the actual 8-bit port. The additional bus cycles generated by the processor to transfer the data are transparent to the programmer, which also allows for fewer lines of code (i.e. one MOVE.L instruction instead of four MOVE.B instructions). This method therfore frees the programmer, but requires external hardware to generate the appropriate DSACK* enc ad handshake.

The second choice, 'pseudo' 32-bit port interfacing, is exactly the opposite. With this method, the system designer has chosen to disallow mis-sized bus transfers. The ex luded dynamic bus sizing circuitry now forces the programmer to interface with the 8-bit peripheral using 8-bit read and write instructions exclusively (i.e. MOVE.B). The deleted hardware from the DSACK generation logic leads to a smaller system chip count and more board space but, causes the programmer to write an increased number of repetitious instructions. Using this method, the peripheral would give a 32-bit port handshake to the processor similar to the memory handshake. The programmer would need to ensure that all data transfers appeared on the proper section of the data bus by generating tyhe appropriate addresses for the peripheral registers.

Generation of Data Strobes

Dynamic bus sizing on the MC68020 requires external hardware to generate the correct data strobes which are often used to enable buffers on the data bus. This is necessary, because the MC68020 drives all sections of the data bus on a write transfer since the processor does not know initially what width of interface port is being used. The hardware designer must ensure that the appropriate section of the data bus is used by either using the generated data strobes in the chip select logic or by using external tri-state bufers on the data bus which are enabled by the generated data strobes.

The logic needed to generate the proper data strobes does not require a large amount of circuitry. One method, shown in Fig 5, can be implemented using a single 24 pin 18-input, 4-output AND-OR-INVERT logic array. This logic array uses as inputs the two least significant address lines and the size information from the processor along with a signal from the decode logic indicating the interface port width. The outputs from this logic array are four separate active low data strobes which can be either be fed back to the decode logic to generate appropriate chip selects, or they can be used to enable tri-state buffers on the system data bus.

The example shown in Fig 5 is for a typical MC68020 processor based system. PALs or FPLAs are suggested by the authors for quick, minimal system designs, but for higher speeds or unique system designs, they can be replaced by a discrete logic implementation such as that shown in Fig 6. This shows that the data strobe generation logic is constructed using three 74LS54 3--2-3 input AND-OR-INVERT gates, one 74LS32 quadruple 2 input OR gate, one 74LS11 triple 3 input AND gate and one 74LS04 hex inverter. Faster TTL families can be substituted if a faster data strobe select is needed in a particular system design.

Interrupt Handling

The interrupt handling capabilities of the MC68020 32-bit microprocessor are similar to those of the other processors in the M68000 family. The processor chip contains three input signal pins labelled as IPL0-IPL2. External hardware is used to set up an encoded interrupt level on these three input pins. The three pins allow 8 different encodings where a code of zero indicates an interrupt level zero, which indicates no interrupt. The other seven possible encodings are used to indicate interrupt levels 1 to 7. When the processor receives an encoded interrupt on these three input pins, it internally compares the incoming level of interrupt with the current level of the interrupt mask - a three bit mask which is held in the processor status register. If the incoming interrupt level is lower than or equal to the current mask level then the processor ignores the interrupt. On the other hand, if the incoming level is higher than the current mask level, the processor internally flags the interrupt as pending and signals the fact via the IPEND output signal. As the processor reaches an instruction execution boundary, it checks for a pending Interrupt Acknowlege (IACK) bus cycle. The IACK cycle begins like a normal bus cycle but the function code output signals are all set to one to indicate CPU address space and the address bus is set indicate an interrupt acknowledge cycle. This bus cycle can be terminated in one of three ways:

a) By the external hardware giving a normal DSACK handshake

b) By the external hardware asserting the Autovector (AVEC) input

c) By the external hardware asserting the Bus error (BERR) input

If the normal DSACK handshake is given, the processor latches the data on the lowest byte of the interface port and uses this 8-bit data as a vector number. If the AVEC input is asserted, the processor internally generates the vector number for the autovector related to the incoming interrupt level. The final option of the BERR handshake forces the processor to internally generate the vector number for the spurious interrupt vector. Once the vector number has been generated, the processor saves the exception vector offset, program counter, and status register on the active supervisor stack and then uses the vector number to go out to the vector table and fetch the 32-bit address' which is the start address of the exception handling routine.

The MC68020 does not contain the synchronous 6800 type bus interface signals found on the earlier processors of the M68000 family, therfore the MC68020 can handle auto-vectored interrupts as fast as normal vectored interrupts. This means that the hardware designer can 'hang' two different interrupt sources on each interrupt level without the need for additional circuitry. One of these sources would generate a vector number during an IACK cycle and the other source would force the processor to autovector. Hence using this technique it is possible to directly support up to 14 external interrupt sources without the need for a large amount of extra priority encoding logic.

Typical Minimum System

By integrating all of the features described in the previous sections,

it is possible to design a typical MC68020 processor based minimum system such as that shown in Fig 7.

In this minimum system both the RAM and ROM memory are implemented as 32-bit wide memory and use the full width of the data bus for operand transfers. The I/O devices are normally only accessed by the processor very infrequently and are therfore designed to use a simple 8-bit wide data bus interface.

The data buffer control logic ensures that the relevant portion(s) of the data bus to the memory and peripheral devices are enabled during every bus cycle initiated by the process. This control logic uses address lines A0 and A1 and the size signals SIZ0 and SIZ1 along with the knowledge of which port size is being used with which portions of the memory map to perform its function. It may be implemented by a PAL or discrete logic as previously described.

The bus handshake logic develops the appropriate DSACK encodings to give to the processor on each bus cycle and also contains the bus error timeout logic. This logic also receives an input from the interrupt logic to prompt it to generate the appropriate handshake for an Interrupt Acknowledge cycle.

The address decode logic is split into two sections. The first section generates the chip select inputs for all of the memory and I/O devices. The second section decodes CPU address space accesses and developes selects for breakpoint instructions, MMU accesses, Coprocessor accesses and Interupt Acknowledge cycles. These generated chip selects could also be used by the bus handshake and buffer control logic to generate the proper DSACK encoding and control the buffers on the data bus.

The interrupt logic takes inputs from all the interrupt sources in the system and generates the encoded three bit interrupt level input to the processor. When the MC68020 runs the Interrupt Acknowledge cycle, this logic also prompts the bus handshake logic to generate either an encoded DSACK, AVEC or BERR handshake.

In the typical minimum system shown in the diagram, the RAM memory is implemented using fast static RAM devices for maximum performance. If a lower performance is acceptable, then dynamic RAM devices could be used in the design. This would require an additional block of circuitry to handle the dynamic RAM refresh requirements which would appear as an additional section in the block diagram of the minimum system.

Conclusions

It has been shown the MC68020 32-bit microprocessor allows for very easy system design because of its simple interface to memory etc. The Dynamic Bus Sizing capabilities even allows the processor to interface with existing memory and I/O systems which may only use an 8 or 16-bit wide interface bus. These features, coupled with the software compatibility of the MC6820 with the other processors of the M68000 family, allow for very high performance processor systems to be quickly developed.

References

1. MacGregor, D. Mothersole, G. and Moyer, B. The Motorola MC68020 IEEE Micro Aug 1984 p101-118
2. Motorola MC68020 32-bit Microprocessor User's Manual Prentice-Hall 1984

List of Figures

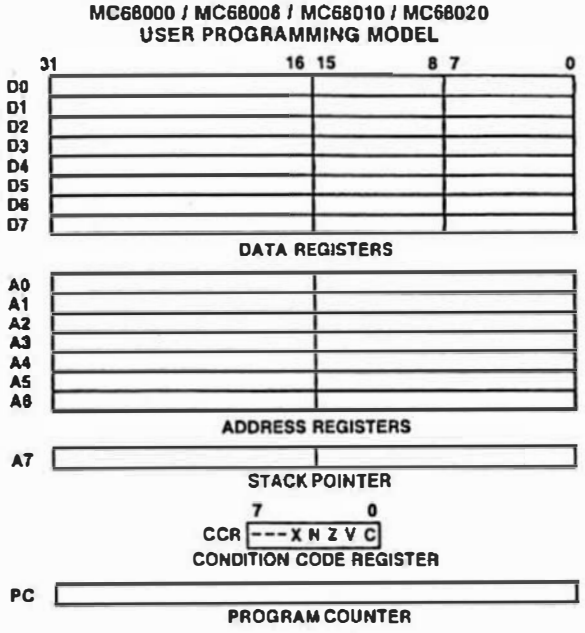


Figure 1

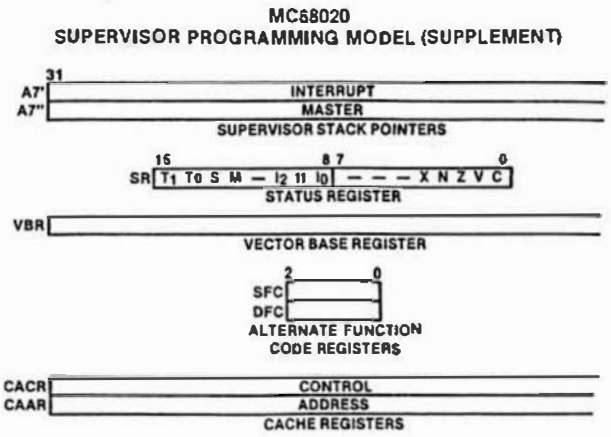


Figure 2

Performance Comparison of the
M68000 Family Processors

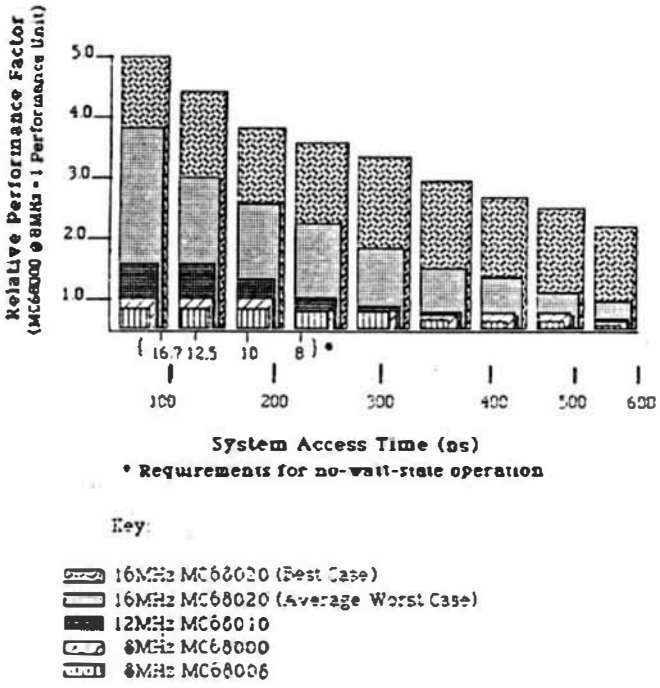
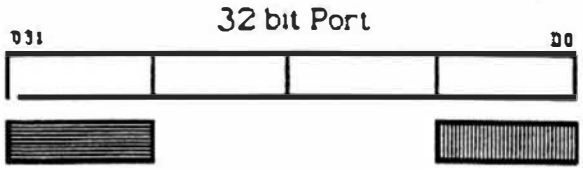


Figure 3



8 bit "True" Port

8 bit "Pseudo" Port

Figure 4

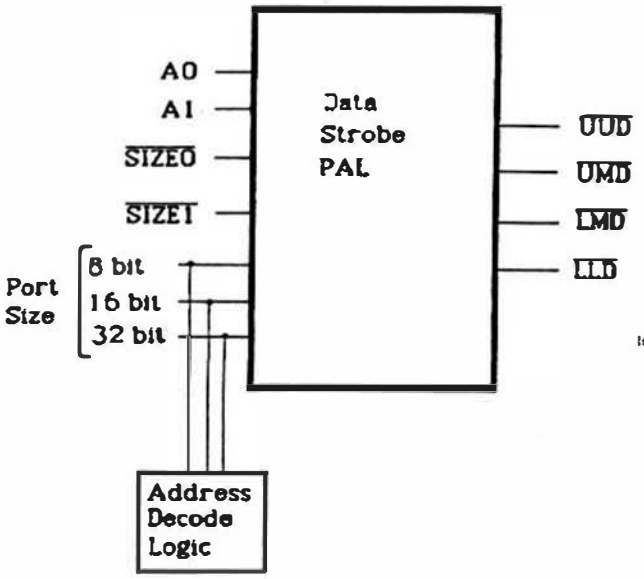


Figure 5

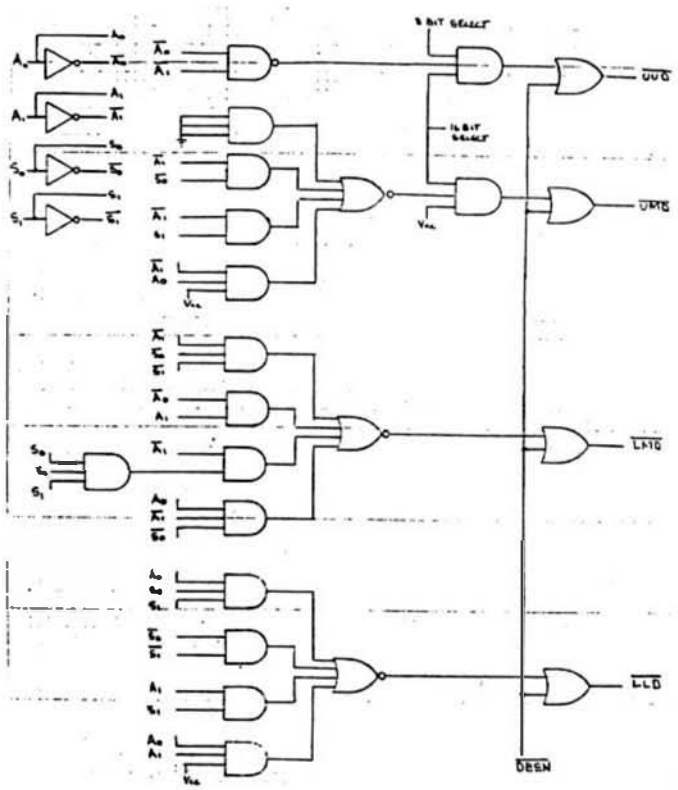


Figure 6

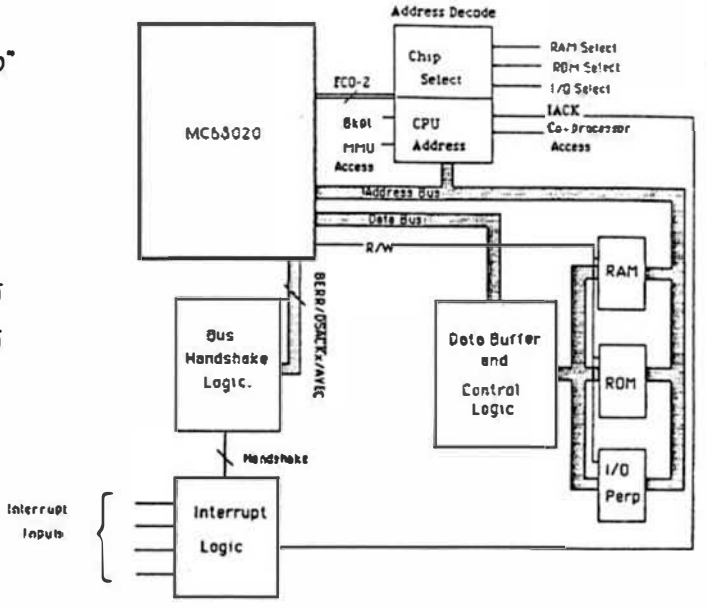
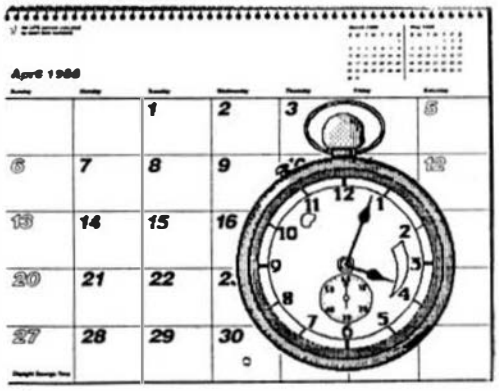


Figure 7

MICROTIME II

Clock/Calendar Board

Clay Cederstrand
326 4th St. South
Martensville, Saskatchewan
S0K 2T0
CANADA



I recently decided to acquire a real time clock/calender board for my homebrew 6809 system running FLEX-09. Peruseing my latest copy of the 68 Micro Journal yielded a couple of possibilities. Since I had built up my micro-computer from bare SS50C and SS30 boards I chose that route again, assembling it myself. The clock/calender card I chose came from the AAA Chicago Computer Company, whom I had purchased hardware/software from before and was quite satisfied with the quality of the product as well as the service.

The board is called the MICROTIME II board, and I purchased it as a bare card for $30 ($45 Canadian). The package was received quite quickly and consisted of the board and a eight page manual. The board is of decent quality, single sided, with no silk screening to indicate component locations or trace paths. The manual, which consists of eight unbound printed pages, does however have a hand drawn diagram to assist you in the correct placement of the components. The rest of the manual contains a schematic, a description of the MSM5832, a description of how to program the chip, and two programs. The first, written in BASIC, allows one to program the MSM5832; the second is a very primative machine language routine that reads the registers and displays them in order on the CRT. Both programs are fine for establishing the workability of the board, but are to primative or awkward to use for reading or writing to the clock under normal circumstances.

The board is very quickly assembled due to the low component count, once together it consists of a 5 volt regulator, the MSM5832, a 6821 PIA, a crystal, and a small number of passive components. The board incorporates battery backup in the form of three size AA nicad batteries which I mounted by glueing inexpensive battery holders onto the board. After assembling the board I made one minor modification to the circuit, I have placed pull up resistors on the address, read, write, and hold pins. The MSM5832 has internal pull down resistors on these pins and the minimum input logic high (1) here is 3.6 volts. Without pull-ups the logic (1) level is only 0.2 volts above the minimum required voltage. To be fair I used the board briefly without the pull up resistors and experienced no problems, but I consider the circuit operation marginal without them. The data lines are open drain and the data sheet also mentions pull-ups here, but the signal levels were always at Vcc or ground during either a read or write so none were added . The pull-ups are easily added to the foil side of the board using a in line 4.7K resistor pack from pin 8 (CS) to pins 2 (WRITE), 3 (READ), 4 thru 7 (A0-A3). The hold pin needs a separate resistor to Vcc or pin eight. There is only one adjustment to be made on the board, and it involves tweaking a small trimmer cap in the crystal circuit to adjust the clock rate.

Once you have the board assembled and operating you will wonder how you managed to get along all this time without one. I have included two assembler programs to read and write to the board. The first, DAYTIME.CMD, reads and displays the day, date, and time in either 12 or 24 hour format. It also puts the correct date into the FLEX date registers. In order to avoid having to enter the date when FLEX boots, one should refer to the FLEX installation manual, page 43 section 12. Specific instructions are issued on how to disable the date prompt. Upon doing this and adding or changing your STARTUP.TXT to include a call to DAYTIME.CMD, one will never have to enter the date again with the added benefit of having the time displayed on bootup. For those who do not have the benefit of the manual for the installation of general FLEX; it gets tougher. The problem is is overlay the call at address $CA02, which is a jump to subroutine , and replace the JSR instruction with a return from sub- routine (RTS). The address to which the JSR instruction points is actually the start of the date routine which you encounter upon booting FLEX. The second assembler routine , SETCLOCK.CMD, allows one to program the clock registers with the day, date, and time in either 12 or 24 hour format.

The board has one additional feature that I do not make use of, and that is the ability to generate interrupts. This is a desirable feature if one wants, for example, to implement print spooling and has no other timer to generate the necessary timed interrupts. The board can be strapped for one of four time intervals as well as the type of interrupt generated, either NMI or a IRQ interrupt. The intervals that are achieveable are roughly 0.3 msec, 1 msec, 16.6 msec, and 1 second. TSC recommends 10 msec for print spooling, so the 16.6 msec interval would work reasonably well.

I have used the board along with the two appended programs and have not had any problems. I have no reservations whatsoever about recommending the board as a inexpensive addition to ones SS50 system. As a last note, the board is no longer advertised as a bare card but comes assembled for $60, my feelings are that this is a good deal, the cost of the parts in addition to the cost of the bare board was only somewhat less than that figure.

- - -

```
*
* The MSM5832 clock driver routine will prompt the user
* for the current time and date information necessary
* to program the clock chip.  The routine is versatile
* in that it gives the user the ability to program the
* clock in either a 12 or 24 hour format, resulting in
* either standard military time or a more conventional
* 12 hour clock, utilizing AM and PM.
*
* Change the CLOCK equate to reflect the position of
* the clock card on the I/O bus.
*
*
* The command syntax is: setclock.cmd
*
* The program authored by Clay Cederstrand
*                         326 4th St. South
*                         Martensville, Sask.
*                         Canada
*                         S0K 2T0
*
*


* The following are entry points for standard
* FLEX routines, I/O and otherwise. This file
* is intended to be used at assembly time and
* called as a library.
*
* Created August 10/84
* Revision date:
*
*flex equates
*
CD03  WARMS   EQU   $CD03     warm start
CD2A  RSTRIO  EQU   $CD2A     restore IO vectors
CD4E  STAT    EQU   $CD4E     set terminal status
CC2B  MEMEND  EQU   $CC2B     end usable memory
CD36  ADDBX   EQU   $CD36     add B to X reg.
*
* Input buffer equates
*
CC14  BUFPNT  EQU   $CC14     address last buffer char
CD27  NXTCH   EQU   $CD27     get next buffer char
CD80  LINBUF  EQU   $CD80     line buffer start to CRFF
*
* Routines to get or output characters
*
CD18  INBUFF  EQU   $CD18     input to line buffer
CD15  GETCHR  EQU   $CD15     get terminal input
CD18  PUTCHR  EQU   $CD18     output char to device
CD1E  PSTRNG  EQU   $CD1E     output a char string
CD09  INCH    EQU   $CD09     get char from terminal
CD24  PCRLF   EQU   $CD24     output a carriage return-line feed
CD39  OUTDEC  EQU   $CD39     output decimal number
CD48  INDEC   EQU   $CD48     input decimal number
CD3C  OUTHEX  EQU   $CD3C     output a hex number
CD45  OUTADR  EQU   $CD45     output hex address
CD21  CLASS   EQU   $CD21     check for alpha-numeric char
CD42  GETHEX  EQU   $CD42     get hex number
*
* DOS routines
*
CD0F  OUTCHR  EQU   $CD0F     output char to device
D406  FMS     EQU   $D406     FMS call
CD3F  RPTERR  EQU   $CD3F     report error
CD2D  GETFIL  EQU   $CD2D     get file spec
CD30  LOAD    EQU   $CD30     loadfile
CD33  SETEXT  EQU   $CD33     set file extension
CD46  DOSCMD  EQU   $CD46     call DOS as subroutine
E060  CLOCK   EQU   $E060
CC0F  FLIDAY  EQU   $CC0F
CC0E  FLIMON  EQU   $CC0E
CC10  FLIYR   EQU   $CC10


*
* Initialize the PIA such that :
* PA0-PA7 are outputs
* PB0-PB7 are outputs
*
```

```
C100                    ORG   $C100
C100 20 01      START   BRA   INIT
C102 01         VER     FCB   1
C103 8E E060    INIT    LDX   #CLOCK
C106 4F                 CLRA
C107 A7 01              STA   1,X       clear control register
C109 A7 03              STA   3,X
C10B 43                 COMA
C10C A7 04              STA   0,X       set A as output
C10E A7 02              STA   2,X       set B as output
C110 86 04              LDA   #%00000100
C112 A7 01              STA   1,X       set data reg active
C114 A7 03              STA   3,X

                *
                * Get the clock data
                *

C116 108E C275  CLKDAT  LDY   #BUFFER
C11A 8E C2A7            LDX   #BANNER
C11D BD CD24            JSR   PCRLF
C120 BD CD1E            JSR   PSTRNG
C123 BD CD24            JSR   PCRLF
C126 8E C377            LDX   #YRMSG    set the year
C129 BD C237            JSR   TTYIN
C12C 8E C390            LDX   #MTHMSG   set the month
C12F BD C237            JSR   TTYIN
C132 8E C3A7            LDX   #DAYMSG   set the date
C135 BD C237            JSR   TTYIN
C138 8E C40E    GETLPR  LDX   #LEAP     is it leap year?
C13B BD CD24            JSR   PCRLF
C13E BD CD1E            JSR   PSTRNG
C141 BD CD15            JSR   GETCHR
C144 8A 20              ORA   #%00100000
C146 81 79              CMPA  #'y       is it?
C148 27 17              BEQ   LEAPYR
C14A 81 6E              CMPA  #'n
C14C 27 08              BEQ   NOLEAP
C14E 8E C4E4            LDX   #ERRMSG
C151 BD CD24            JSR   PCRLF
C154 BD CD1E            JSR   PSTRNG
C157 20 DF              BRA   GETLPR
C159 A6 A3      NOLEAP  LDA   ,--Y
C15B 84 03              ANDA  #%00000011 set D10 for no leap year
C15D A7 A1              STA   ,Y++
C15F 20 06              BRA   GETDAY
C161 A6 A3      LEAPYR  LDA   ,--Y
C163 8A 04              ORA   #%00000100 set D10 for leap year
C165 A7 A1              STA   ,Y++
C167 8E C3BD    GETDAY  LDX   #WATDAY
C16A BD C237            JSR   TTYIN     set the day of week
C16D A6 A2              LDA   0,-Y
C16F 31 3F              LEAY  -1,Y
C171 A7 A0              STA   0,Y+
C173 8E C424            LDX   #HRMSG    set the hour
C176 BD C237            JSR   TTYIN
C179 8E C476    GETCLK  LDX   #WATCLK   set 12/24 format
C17C BD CD24            JSR   PCRLF
C17F BD CD1E            JSR   PSTRNG
C182 BD CD15            JSR   GETCHR
C185 8A 20              ORA   #%00100000
C187 81 6E              CMPA  #'n
C189 27 40              BEQ   ITS24
C18B 81 79              CMPA  #'y
C18D 27 08              BEQ   GETAM
C18F 8E C4E4            LDX   #ERRMSG
C192 BD CD1E            JSR   PSTRNG
C195 BD CD24            JSR   PCRLF
C198 20 DF              BRA   GETCLK
C19A 8E C454    GETAM   LDX   #AMPMSG   set AM or PM
C19D BD CD24            JSR   PCRLF
C1A0 BD CD1E            JSR   PSTRNG
C1A3 BD CD15            JSR   GETCHR
C1A6 8A 20              ORA   #%00100000
C1A8 81 79              CMPA  #'y
C1AA 27 17              BEQ   ITSPM
C1AC 81 6E              CMPA  #'n
C1AE 27 08              BEQ   ITSAM
C1B0 8E C4E4            LDX   #ERRMSG
C1B3 BD CD24            JSR   PCRLF
C1B6 BD CD1E            JSR   PSTRNG
C1B9 20 DF              BRA   GETAM
C1BB A6 A3      ITSAM   LDA   ,--Y      set H10 and set 12/AM
C1BD 84 03              ANDA  #%00000011
C1BF A7 A1              STA   ,Y++
C1C1 20 0E              BRA   GETMIN
C1C3 A6 A3      ITSPM   LDA   ,--Y
C1C5 8A 04              ORA   #%00000100 set H10 for 12/PM
C1C7 A7 A1              STA   ,Y++
C1C9 20 06              BRA   GETMIN
C1CB A6 A3      ITS24   LDA   ,--Y
C1CD 8A 08              ORA   #%00001000
C1CF A7 A1              STA   ,Y++
C1D1 8E C4A1    GETMIN  LDX   #MINMSG
C1D4 BD C237            JSR   TTYIN
C1D7 8E C280    ALL_OK  LDX   #OK
C1DA BD CD24            JSR   PCRLF
C1DD BD CD1E            JSR   PSTRNG
C1E0 BD CD15            JSR   GETCHR
C1E3 8A 20              ORA   #%00100000
C1E5 81 79              CMPA  #'y
C1E7 1027 FF2B          LBEQ  CLKDAT
C1EB 81 6E              CMPA  #'n
C1ED 27 08              BEQ   SETHLD
C1EF 8E C4E4            LDX   #ERRMSG
C1F2 BD CD24            JSR   PCRLF
C1F5 BD CD1E            JSR   PSTRNG
C1F8 20 80              BRA   ALL_OK

                *
                * The following routine enables the hold function of
                * the clock chip, then proceeds to write the clock
                * registers from the top (YR10) to the last (MI1).
                * The seconds register is automatically reset to 0
                * upon writing into the chip.
                *

C1FA 8E E060    SETHLD  LDX   #CLOCK
C1FD 86 2C              LDA   #$2C      set hold plus address Y10
C1FF 1F 89              TFR   A,B
C201 A7 02              STA   2,X
C203 4A         LOOP    DECA
C204 81 00              CMPA  #$00
C206 26 FB              BNE   LOOP
C208 108E C275          LDY   #BUFFER
C20C E7 02      SETWRT  STB   2,X
C20E A6 A0              LDA   0,Y+
C210 A7 84              STA   0,X
C212 CA 40              ORB   #%01000000
C214 E7 02              STB   2,X       set hold,write,and address
C216 C4 2F              ANDB  #%00101111 strobe write
C218 E7 02              STB   2,X
C21A 5A                 DECB
C21B C1 21              CMPB  #$21      check for last register
C21D 27 02              BEQ   DONE
C21F 20 EB              BRA   SETWRT
C221 8E C4C5    DONE    LDX   #FINISH
C224 BD CD24            JSR   PCRLF
C227 BD CD1E            JSR   PSTRNG
C22A BD CD15            JSR   GETCHR
C22D 4F                 CLRA
C22E A7 02              STA   2,X       set all control lines low
C230 A7 84              STA   0,X       set all data lines low
C232 7F C003            JMP   $C003
C235 31 3F      TTYSET  LEAY  -1,Y
C237 BF C273    TTYIN   STX   TEMP
C23A BD CD24            JSR   PCRLF
C23D BD CD1E            JSR   PSTRNG
C240 BD CD15            JSR   GETCHR
C243 BD 12              BSR   STRIP
C245 C1 00              CMPB  #$00
C247 27 EE              BEQ   TTYIN
C249 A7 A0              STA   0,Y+
C24B BD CD15            JSR   GETCHR
C24E BD 07              BSR   STRIP
C250 C1 00              CMPB  #$00      check for entry error
C252 27 E1              BEQ   TTYSET
C254 A7 A0              STA   0,Y+
C256 39                 RTS
```

```
                        *
                        * This routine masks the high order nybble
                        *
C257 C6  FF     STRIP   LDB   #$FF
C259 81  30             CMPA  #$30      check for lower than 0
C25B 2D  07             BLT   ERROR
C25D 81  39             CMPA  #$39      check for greater than 9
C25F 2E  03             BGT   ERROR
C261 84  0F             ANDA  #$0F
C263 39                 RTS
C264 C6  00     ERROR   LDB   #$00
C266 8E  C4E4           LDX   #MSGMSG
C269 BD  CD24           JSR   PCRLF
C26C BD  CD1E           JSR   PSTRNG
C26F BE  C273           LDX   TEMP
C272 39                 RTS

C273 0000       TEMP    FDB   0
C275            BUFFER  RMB   11
```

```
                        *
                        * Message Table
                        *
C280 48 61 76 65 OK     FCC   /Have you made any entry errors (Y\N) ?/,$4
C284 20 79 6F 75
C288 20 6D 61 64
C28C 65 20 61 6E
C290 79 20 65 6E
C294 74 72 79 20
C298 65 72 72 6F
C29C 72 73 20 28
C2A0 59 5C 4E 29
C2A4 20 3F 04
C2A7 41 6C 6C 20 ERMSG  FCC   /All entrys should be two digits, with a leading/,$0,$A
C2AB 65 6E 74 72
C2AF 79 73 20 73
C2B3 68 6F 75 6C
```

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

Telex 5106006630

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

K-BASIC now makes the multitude of TSC MBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and BINGO -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (os-9 or FLEX), including the Assembler

!!! Special !!! ~~$109.00~~ $ 99.49

SAVE $100.00

SCULPTOR

Sculptor for 68020 OS-9 and UniFLEX $995

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. PLUS, the Application can also be run on MS-DOS and Unix machines! Sculptor handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes ISAM File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for Development Package/Run-time. Discounts available for purchases of 5 or more Run-time Packages.

OS-9 / UniFLEX --			68000 UniFLEX --			MS DOS --	
IBM PC Zenix --	$995.00/$175.00		Altos Zenix --	$1595.00/$265.00		--	$595.00/$115.00
MS DOS Network --	*	**	UNIX --	*	**	PC DOS --	* **

* Full Developement Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

!!! Special Buy Out !!! SPECIAL - Limited Quantity !!! Special Buy Out !!!

6809 FLEX SOFTWARE

TSC Flex Utilities	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Flex Precompiler	was $50.00	Now only $35.00
TSC Flex Basic	was $75.00	Now only $50.00
TSC Flex Diagnostic	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Assembler	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $75.00	Now only $50.00

DISKS ONLY

TSC Extended Precompiler	was $50.00	Now only $35.00
TSC Basic Flex	was $75.00	Now only $50.00
TSC Diagnostics	was $75.00	Now only $50.00
TSC Utilities	was $75.00	Now only $50.00

MANUALS ONLY

TSC Basic Precompiler	was $25.00	Now only $20.00
TSC Text Editor	was $25.00	Now only $20.00
TSC Mnemonic Assembler	was $25.00	Now only $20.00

DISKS ONLY - 6800 Software

TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $100.00	Now only $70.00
TSC Assemblers	was $50.00	Now only $35.00

MANUALS ONLY - 6800 Software

TSC Diagnostics	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Text Processor	was $25.00	Now only $20.00

!!! Please Specify Your Operating System & Disk Size !!!

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the TSC **Macro Assembler**. F, CCF - $99.95

Macro Assembler for TSC -- The FLEX **STANDARD** Assembler.
Special -- CCF $35.00; F $50.00

OSM Extended 6809 Macro Assembler from **Lloyd I/O**. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 $99.00

Relocating Assembler w/Linking Loader from **TSC**. -- Use with many of the C and Pascal Compilers. F,CCF $150.00

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - $90.00

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** --
Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - $50.00 Source - $50.00 each
any 3 - $100.00
complete set w/ C Source (except the 68000 Source) -$300.00

XASM Cross Assemblers for FLEX from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - $150.00

CRASM from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- $399.00

CRASM 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000**.
FLEX, CCF, OS-9 $249.00

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCO (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide. F, CCF - $198.00

WHIMSICAL from **Whimsical Developments** -- Now supports **Real Numbers**. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - $195.00

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - $295.00

C Compiler from **Introl** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, CCF, OS-9 (Level II **ONLY**), U - $575.00

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$99.95** F 8" **$99.95**

PASCAL Compiler from **Compusoft** (now **Certified Software**) -- For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Assb. and Linking Loader.
F and CCF - $425.00 One Year Maint. - $100.00

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC compatible**. The compiler compiles to Assembly Language Source Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - **$199.00**

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- **$99.95**

FORTH from **Stearns Electronics** -- A CoCo FORTH Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - $90.95

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

GENERAL

SOFTWARE DEVELOPMENT

BasicO9 XRef from Southeast Media -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

F, CCF --- **RACE Utility** 5" - $40.00, 8" - $50.00

DUB from Southeast Media -- A UniFLEX "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled basic** Programs. Works w/ ALL Versions of 680g UniFLEX basic. U - $219.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- **TSC Extended BASIC** program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 VDisk from Southeast Media -- For **Level I only.** Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format OAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from Southeast Media -- Written in **BASICO9** (with Source), includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an **OS-9** disk to **FLEX** Format so it can be used normally by FLEX; and FLEX, a BASICO9 Program that does the actual **read or write** function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** O-$39.95

COPYMULT from Southeast Media -- Copy LARGE Disks to several smaller disks. FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB OOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.
F and CCF 5" - $90.00 F 8" - $65.00

FLEX DISK UTILITIES from Computer System Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test Disk** for errors; **Compare** two **Disks**; a fast **Disk Backup** Program; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with **EXTRAS** over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare, Merge,** or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare two **unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC **Cross-Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC, XBASIC,** and **[illegible] BASIC** Programs.
ALL Utilities **Include Source** (either BASIC or A.L. Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CHODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $90.00

XDATA from Southeast Media -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 **Chess Program from Southeast Media** -- Requires FLEX and Displays on **Any** Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

VISA

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

BUSINESS

WORD PROCESSING

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES ---> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES ---> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES ---> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:

Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features some of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95

SPELL8 "Computer Dictionary" from Southeast Media -- OVER 120,000 words! Look up a word from within your Editor or Word Processor (with the SPW.CMD Utility which operates in the FLEX IOCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of Small Disk Storage systems.

!! SPECIAL LIMITED TIME OFFER !! F and CCF - $99.95

DATA BASE - ACCOUNTING

XDMS from Westchester Applied Business Systems -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENERATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95

ACCOUNTING PACKAGES -- Great Plains Computer Co. and Universal Data Research, Inc. both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from Computer Systems Center -- Electronic Spread Sheet for the 6809.

F, SPECIAL CCF and OS9 - $200.00, U - $395.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. KeePs item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

```
C2B7 64 20 62 65
C2BB 20 74 77 6F
C2BF 20 64 69 67
C2C3 69 74 73 2C
C2C7 20 77 69 74
C2CB 68 20 61 20
C2CF 6C 65 61 64
C2D3 69 6E 67 00
C2D7 0A
C2D8 7A 65 72 6F          FCC   /zero mandatory on the day of week,
                                and days and months/,$D,$4

C2DC 20 6D 61 6E
C2E0 64 61 74 6F
C2E4 72 79 20 6F
C2E8 6E 20 74 68
C2EC 65 20 64 61
C2F0 79 20 6F 66
C2F4 20 77 65 65
C2F8 68 2C 20 61
C2FC 6E 64 20 64
C300 61 79 73 20
C304 61 6E 64 20
C308 6D 6F 6E 74
C30C 68 73 00 0A
C310 74 68 61 74          FCC   /that are less than 10. Answers
                                requiring (Y)es or (N)o/,$D,$A

C314 20 61 72 65
C318 20 6C 65 73
C31C 73 20 74 68
C320 61 6E 20 31
C324 30 2E 20 41
C328 6E 73 77 65
C32C 72 73 20 72
C330 65 71 75 69
C334 72 69 6E 67
C338 20 28 59 29
C33C 65 73 20 6F
C340 72 20 28 4E
C344 29 6F 00 0A
C348 63 61 6E 20          FCC   /can be either upper or lowercase
                                Y's or n's./,$D,$A,$4

C34C 62 65 20 65
C350 69 74 68 65
C354 72 20 75 70
C358 70 65 72 20
C35C 6F 72 20 6C
C360 6F 77 65 72
C364 63 61 73 65
C368 20 79 27 73
C36C 20 6F 72 20
C370 6E 27 73 2E
C374 00 0A 04
C377 54 68 65 20  YRMSG   FCC   /The current year is : 19/,$4
C378 63 75 72 72
C37F 65 6E 74 20
C383 79 65 61 72
C387 20 69 73 20
C388 3A 20 31 39
C38F 04
C390 54 68 65 20  MONMSG  FCC   /The current month is :/,$4
C394 63 75 72 72
C398 65 6E 74 20
C39C 60 6F 6E 74
C3A0 68 20 69 73
C3A4 20 3A 04
C3A7 54 68 65 20  DAYMSG  FCC   /The current date is :/,$4
C3A8 63 75 72 72
C3AF 65 6E 74 20
C3B3 64 61 74 65
C3B7 20 69 73 20
C3B8 3A 04
C3BD 57 69 74 68  WATDAY  FCC   /With SUNDAY as 00 thru SATURDAY as 06/,$D,$A
C3C1 20 53 55 4E
C3C5 44 41 59 20
C3C9 61 73 20 30
C3CD 30 20 74 68
C3D1 72 75 20 53
C3D5 41 54 55 52
C3D9 44 41 59 20
C3DD 61 73 20 30
C3E1 36 0D 0A
```

```
C3E4 45 6E 74 65          FCC   /Enter the number of the day of the week :/,$4
C3E8 72 20 74 68
C3EC 65 20 6E 75
C3F0 60 62 65 72
C3F4 20 6F 66 20
C3F8 74 68 65 20
C3FC 64 61 79 20
C400 6F 66 20 74
C404 68 65 20 77
C408 65 65 68 20
C40C 3A 04
C40E 49 73 20 74  LEAP    FCC   /Is this a leap year ?/,$4
C412 68 69 73 20
C416 61 20 6C 65
C41A 61 70 20 79
C41E 65 61 72 20
C422 3F 04
C424 54 68 65 20  HRMSG   FCC   /The hour ( in either 12 or 24 hour format ) is:/,$4
C428 68 6F 75 72
C42C 20 28 20 69
C430 6E 20 65 69
C434 74 68 65 72
C438 20 31 32 20
C43C 6F 72 20 32
C440 34 20 68 6F
C444 75 72 20 66
C448 6F 72 6D 61
C44C 74 20 29 20
C450 69 73 3A 04
C454 49 73 20 74  AMPMSG  FCC   /Is the time of day currently PM ?/,$4
C458 68 65 20 74
C45C 69 6D 65 20
C460 6F 66 20 64
C464 61 79 20 63
C468 75 72 72 65
C46C 6E 74 6C 79
C470 20 50 4D 20
C474 3F 04
C476 44 6F 20 79  WATCLK  FCC   /Do you wish a 12 hour AM\PM format clock ?/,$4
C47A 6F 75 20 77
C47E 69 73 68 20
C482 61 20 31 32
C486 20 68 6F 75
C48A 72 20 41 40
C48E 5C 50 4D 20
C492 66 6F 72 6D
C496 61 74 20 63
C49A 6C 6F 63 68
C49E 20 3F 04
C4A1 54 68 65 20  MINMSG  FCC   /The minute following this one is ?:/,$4
C4A5 60 69 6E 75
C4A9 74 65 20 66
C4AD 6F 6C 6C 6F
C4B1 77 69 6E 67
C4B5 20 74 68 69
C4B9 73 20 6F 6E
C4BD 65 20 69 73
C4C1 20 3F 3A 04
C4C5 48 69 74 20  FINISH  FCC   /Hit any key to start the clock/,$4
C4C9 61 6E 79 20
C4CD 6B 65 79 20
C4D1 74 6F 20 73
C4D5 74 61 72 74
C4D9 20 74 68 65
C4DD 20 63 6C 6F
C4E1 63 6B 04
C4E4 59 6F 75 20  ERRMSG  FCC   /You have made a incorrect entry,
                                Please do correctly /,$4

C4E8 68 61 76 65
C4EC 20 6D 61 64
C4F0 65 20 61 20
C4F4 69 6E 63 6F
C4F8 72 72 65 63
C4FC 74 20 65 6E
C500 74 72 79 2C
C504 20 70 6C 65
C508 61 73 65 20
C50C 64 6F 20 63
C510 6F 72 72 65
C514 63 74 6C 79
C518 20 04
                  END   START
```

```
0 ERROR(S) DETECTED

SYMBOL TABLE:

ABORT  CD36  ALL_OK C107  ANMSG  C454  BANNER C2A7  BUFFER C275
BUFPNT CC14  CLASS  CD21  CLKDAT C116  CLOCK  E060  DAYMSG C3A7
DDCMND CD48  DONE   C221  ERRMSG C4E4  ERROR  C26A  FINISH C4C5
FLIDAY CC0F  FLIMON CC0E  FLIYR  CC10  FMS    D406  GETAM  C19A
GETCHR CD15  GETCLK C179  GETDAY C167  GETFIL CD2D  GETHEX CD42
GETLPR C138  GETMIN C1D1  HRMSG  C424  INBUFF CD18  INCH  CD09
INDEC  CD48  INIT   C103  ITS24  C1C8  ITSAM  C1B8  ITSPM  C1C3
LEAP   C40E  LEAPYR C161  LINBUF CC80  LOAD   CD30  LOOP  C203
MEMEND CC2B  MINMSG C4A1  MONMSG C390  NOLEAP C159  NOTCH CD27
OK     1280  OUTADR CD45  OUTCHR CD0F  OUTDEC CD39  OUTHEX CD3C
PCRLF  CD24  PSTRNG CD1E  PUTCHR CD1B  RPTERR CD3F  RSTRIO CD2A
SETEXT CD33  SETHLD C1FA  SETMRT C20C  START  C100  STAT  CD4E
STRIP  C257  TEMP   C273  TTYDET C235  TTYIN  C237  VER   C102
WARMS  CD03  WATCLK C476  WATDAY C380  YRMSG  C377
```

```
*
* This clock read program will read the registers of
* a MM58167 real time clock/calender chip. It will
* display the time in either 12 or 24 hour format.
* If the clock is programmed as a 12 hour clock than
* AM or PM will be displayed with the time.
*
* The command syntax is: time.cmd
*
* The program authored by Clay Cederstrand
*                         326 4th St South
*                         Martensville, Sask.
*                         Canada
*                         S0K 2T0
*
*

* The following are entry points for standard
* FLEX routines, I/O and otherwise. This file
* is intended to be used at assembly time and
* called as a library.
*
* Created August 10/84
* Revision date:
*
*Flex equates
*
CD03  WARMS  EQU  $CD03    warm start
CD2A  RSTRIO EQU  $CD2A    restore IO vectors
CD4E  STAT   EQU  $CD4E    get terminal status
CC2B  MEMEND EQU  $CC2B    end usable memory
CD36  ADDBX  EQU  $CD36    add B to X reg.
*
* Input buffer equates
*
CC14  BUFPNT EQU  $CC14    address last buffer char
CC27  NXTCH  EQU  $CC27    get next buffer char
CC80  LINBUF EQU  $CC80    line buffer start to COFF
*
* Routines to get or output characters
*
CD18  INBUFF EQU  $CD18    input to line buffer
CD15  GETCHR EQU  $CD15    get terminal input
CD1B  PUTCHR EQU  $CD1B    output char to device
CD1E  PSTRNG EQU  $CD1E    output a char string
CD09  INCH   EQU  $CD09    get char from terminal
CD24  PCRLF  EQU  $CD24    output a carriage return-line feed
CD39  OUTDEC EQU  $CD39    output decimal number
CD48  INDEC  EQU  $CD48    input decimal number
CD3C  OUTHEX EQU  $CD3C    output a hex number
CD45  OUTADR EQU  $CD45    output hex address
CD21  CLASS  EQU  $CD21    check for alpha-numeric char
CD42  GETHEX EQU  $CD42    get hex number
*
* DOS routines
*
CD0F  OUTCHR EQU  $CD0F    output char to device
D406  FMS    EQU  $D406    FMS call
CD3F  RPTERR EQU  $CD3F    report error
CD2D  GETFIL EQU  $CD2D    get file spec
CD30  LOAD   EQU  $CD30    load file
CD33  SETEXT EQU  $CD33    set file extension
CD48  DDCMND EQU  $CD48    call DOS as subroutine
E060  CLOCK  EQU  $E060
CC0F  FLIDAY EQU  $CC0F
CC0E  FLIMON EQU  $CC0E
CC10  FLIYR  EQU  $CC10

*
* Initialize the PIA as follows:
* PA0-PA7 as inputs
* PB0-PB7 as outputs
*

C100                    ORG   $C100
C100 20 01      START   BRA   INIT
C102 01         VER     FCB   1
C103 8E E060    INIT    LDX   #CLOCK
C106 4F                 CLRA
C107 A7 01              STA   1,X       INITIALIZE BOTH PORTS
C109 A7 03              STA   3,X
C10B A7 84              STA   0,X       SET A AS INPUT
C10D 43                 COMA
C10E A7 02              STA   2,X       SET B AS OUTPUT
C110 86 04              LDA   #%00000100
C112 A7 01              STA   1,X       SET DATA REG
C114 A7 03              STA   3,X
C116 86 20              LDA   #$20
C118 A7 02              STA   2,X       ENABLE CLOCK HOLD
C11A 7F C22A            CLR   AMFLG
C11D 7F C229            CLR   MILCLK
C120 4A         SETHLD  DECA
C121 81 00              CMPA  #$00      DELAY .150ms
C123 26 FB              BNE   SETHLD
*
* READ THE CLOCK REGISTERS
*
C125 108E C21D          LDY   #REGVAL
C129 86 3C              LDA   #$3C      SET HOLD,READ,ADDRESS
C12B B7 E062    RDCLK   STA   CLOCK+2
C12E 12                 NOP             DELAY FOR READ
C12F F6 E060            LDB   CLOCK
C132 C4 0F              ANDB  #$0F      MASK HIGH ORDER BITS
C134 81 35              CMPA  #$35      CHECK FOR HR10
C136 27 09              BEQ   CHK24     12 OR 24 HR CLOCK?
C138 E7 A0      STRVAL  STB   0,Y+
C13A 4A                 DECA
C13B 81 2F              CMPA  #$2F
C13D 26 EC              BNE   RDCLK     NOT DONE YET
C13F 20 1C              BRA   CLRHLD
*
* CHECK FOR 12 OR 24 HOUR CLOCK
*
C141 34 04      CHK24   PSHS
C143 C4 08              ANDB  #%00001000
C145 27 02              BEQ   CHKAM     HOT 24 HR CLOCK
C147 20 0E              BRA   BAMODN
C149 73 C22B    CHKAM   COM   MILCLK
C14C 35 04              PULS
C14E 34 04              PSHS
C150 C4 04              ANDB  #%00000100 CHECK FOR AM
C152 27 03              BEQ   BAMODN
C154 73 C22A            COM   AMFLG
C157 35 04      BAMODN  PULS
C159 C4 03              ANDB  #$03
C15B 20 DB              BRA   STRVAL
C15D 4F         CLRHLD  CLRA
C15E B7 E062            STA   CLOCK+2
C161 108E C21D          LDY   #REGVAL
C165 A6 A4      STRFLX  LDA   ,Y
C167 BD C214            JSR   ASCII
C16A B7 C304            STA   YR10      STORE ASCII YR10
C16D A6 A0              LDA   0,Y+      GET YR10 AGAIN
C16F BD C217            JSR   MAKBIN    MAKE BINARY NUMBER
C172 F7 CC10            STB   FLIYR
C175 A6 A0              LDA   0,Y+      GET YR1
C177 BD C214            JSR   ASCII
C17A B7 C305            STA   YR1
C17D A6 A0              LDA   0,Y+      GET MO10
C17F BD C217            JSR   MAKBIN
C182 F7 CC0E            STB   FLIMON
```

```
C185 5A                 DECB
C186 8E  C272           LDX   #MONTBL
C189 86  0A             LDA   #10
C18B 3D                 MUL
C18C 3A                 ABX
C18D 34  20             PSH   Y
C18F 108E C2F5          LDY   #DAYMSG
C193 A6  80     STRMON  LDA   0,X+
C195 81  30             CMPA  #$30
C197 27  04             BEQ   DONEMO
C199 A7  A0             STA   0,Y+
C19B 20  F6             BRA   STRMON
C19D 35  20     DONEMO  PUL   Y
C19F 31  21             LEAY  1,Y
C1A1 A6  A4             LDA   0,Y       GET DIG
XC1A3 BD C214           JSR   ASCII
C1A6 B7  C2FF           STA   DIG
C1A9 A6  A0             LDA   0,Y+      GET DIG AGAIN
XC1AB BD C217           JSR   MAKBIN
C1AE F7  CC06           STB   FLIDAY

XC1B3 BD C214           JSR   ASCII
C1B6 B7  C300           STA   DI
                *
                * GET WEEKDAY
                *
C1B9 A6  A0             LDA   0,Y+      GET WKDAY
C1BB 8E  C22C           LDX   #DAYTBL
C1BE C6  0A             LDB   #10
C1C0 3D                 MUL
C1C1 3A                 ABX

C1C2 34  20             PSH   Y
C1C4 108E C2EA          LDY   #DAYMSG
C1C8 A6  80     STRDAY  LDA   0,X+
C1CA 81  30             CMPA  #$30
C1CC 27  04             BEQ   DOREST
C1CE A7  A0             STA   0,Y+
C1D0 20  F6             BRA   STRDAY
C1D2 C6  04     DOREST  LDB   #4
C1D4 35  20             PUL   Y
C1D6 8E  0B0E           LDX   #HR10
C1D9 A6  A0     DONEDY  LDA   0,Y+      GET HR10
XC1DB BD C214           JSR   ASCII
C1DE A7  80             STA   0,X+
C1E0 5A                 DECB
C1E1 27  0E             BEQ   DONTIM
C1E3 C1  04             CMPB  #$04
C1E5 26  02             BNE   CHEK2
C1E7 30  01             LEAX  1,X
C1E9 C1  02     CHEK2   CMPB  #$02
C1EB 26  02             BNE   NOCOME
C1ED 30  01             LEAX  1,X
C1EF 20  E8     NOCOME  BRA   DONEDY
C1F1 B6  C228   DONTIM  LDA   MILCLK
C1F4 27  12             BEQ   ITS24
C1F6 8E  C317           LDX   #AMPM
C1F9 86  41             LDA   #'A
C1FB F6  C228           LDB   AMFLG
C1FE 27  02             BEQ   ITSAM
C200 86  50             LDA   #'P
C202 A7  80     ITSAM   STA   0,X+
C204 86  4D             LDA   #'M
C206 A7  80             STA   0,X+
C208 BD CD24    ITS24   JSR   PCRLF
C20B 8E  C2EA           LDX   #DAYMSG
C20E BD CD1E            JSR   PSTRNG
C211 7E  CD03           JMP   WARMS
                *
                * CREATE ASCII DIGIT
                *
C214 8A  30     ASCII   ORA   #$30
C216 39                 RTS
                *
                * MAKE A BINARY NUMBER
                * FOR FLEX
                *
C217 C6  0A     MAKBIN  LDB   #10
C219 3D                 MUL
C21A EB  A4             ADDB  0,Y
C21C 39                 RTS

                *
                * TABLES
                *
C21D            HRTVAL  RMB   13
C22A            AMFLG   RMB   1
C22B            MILCLK  RMB   1
                *
                * Daytable messages
                *
C22C 53 55 4E 44 DAYTBL FCC   /SUNDAY0  /
C230 41 59 30 20
C234 20 20
C236 4D 4F 4E 44        FCC   /MONDAY0  /
C23A 41 59 30 20
C23E 20 20
C240 54 55 45 53        FCC   /TUESDAY0 /
C244 44 41 59 30
C248 20 20
C24A 57 45 44 4E        FCC   /WEDNESDAY0/
C24E 45 53 44 41
C252 59 30
C254 54 48 55 52        FCC   /THURSDAY0 /
C258 53 44 41 59
C25C 30 20
C25E 46 52 49 44        FCC   /FRIDAY0  /
C262 41 59 30 20
C266 20 20
C268 53 41 54 55        FCC   /SATURDAY0 /
C26C 52 44 41 59
C270 30 20
C272 4A 41 4E 55 MONTBL FCC   /JANUARY0 /
C276 41 52 59 30
C27A 20 20
C27C 46 45 42 52        FCC   /FEBRUARY0 /
C280 55 41 52 59
C284 30 20
C286 4D 41 52 43        FCC   /MARCH0   /
C28A 48 30 20 20
C28E 20 20
C290 41 50 52 49        FCC   /APRIL0   /
C294 4C 30 20 20
C298 20 20
C29A 4D 41 59 30        FCC   /MAY0     /
C29E 20 20 20 20
C2A2 20 20
C2A4 4A 55 4E 45        FCC   /JUNE0    /
C2A8 30 20 20 20
C2AC 20 20
C2AE 4A 55 4C 59        FCC   /JULY0    /
C2B2 30 20 20 20
C2B6 20 20
C2B8 41 55 47 55        FCC   /AUGUST0  /
C2BC 53 54 30 20
C2C0 20 20
C2C2 53 45 50 54        FCC   /SEPTEMBER0/
C2C6 45 4D 42 45
C2CA 52 30
C2CC 4F 43 54 4F        FCC   /OCTOBER0 /
C2D0 42 45 52 30
C2D4 20 20
C2D6 4E 4F 56 45        FCC   /NOVEMBER0 /
C2DA 4D 42 45 52
C2DE 30 20
C2E0 44 45 43 45        FCC   /DECEMBER0 /
C2E4 4D 42 45 52
C2E8 30 20

                *
                * MESSAGE
                *
C2EA 20 20 20 20 DAYMSG FCC   /       /,$D,$A
C2EE 20 20 20 20
C2F2 20 0D 0A
C2F5 20 20 20 20 AMPMSG FCC   /       /
C2F9 20 20 20 20
C2FD 20 20
C2FF 00         DIG     FCB   0
C300 00         DI      FCB   0
C301 2C         COMMA   FCC   /,/
C302 31 39      YEAR    FCC   /19/
C304 00         YR10    FCB   0
C305 00         YR1     FCB   0
```

```
C306 20 00 0A      EOL     FCC   / /,$0,$A
C309 54 49 4D 45   TIME    FCC   /TIME /
C300 20
C30E 00            HR10    FCB   0
C30F 00            HR1     FCB   0
C310 3A            COLON   FCC   /:/
C311 00            MI10    FCB   0
C312 00            MI1     FCB   0
C313 3A                    FCC   /:/
C314 00            S10     FCB   0
C315 00            S1      FCB   0
C316 20            SPA     FCC   / /
C317 20 20 00 0A   AMPMSG  FCC   /  /,$0,$A,4
C318 04
                           END   START

0 ERROR(S) DETECTED
```

```
SYMBOL TABLE:
ADDBX  C036  AMFLG  C22A  AMPMSG C317  ASCII  C214  BAUDON C157
BUFPNT CC14  CHEK2  C1E9  CHK24  C141  CHKAA  C149  CLASS  CD21
CLOCK  B060  CLRHLD C150  COLON  C310  COMMA  C301  D1     C300
D10    C2FF  DAYMSG C2EA  DAYTBL C22C  DDCMND CD4B  DDMEDY C109
DDMMYD C191  DONTIM C1F1  DOREST C102  EOL    C306  FLXDAY CD0F
FLXMON CD0E  FLXYR  CC10  FMS    D406  GETCHR CD15  GETFIL CD2D
GETNET CD42  HR1    C30F  HR10   C30E  INBUFF CD1B  INCH   CD09
JMDEC  CD48  INIT   C103  ITS20  C208  ITSAM  C202  LINBUF CD80
LOAD   CD30  MAKBIN C217  MEMEND CC2B  MI1    C312  MI10   C311
MILCLK C22B  MONMSG C2F5  MONTBL C272  MODDNE C1EF  NOTCH  CD27
OUTADR CD45  OUTCHR CD0F  OUTDEC CD89  OUTHEX CD3C  PORLF  CD24
PSTRNG CD1E  PUTCHR CD18  RDLLK  C12B  REGVAL C21D  RPTERR CD3F
RSTR10 CD2A  S1     C315  S10    C314  SETEXT CD33  SETHLD C120
SPA    C316  START  C100  STAT   CD4E  STRDAY C1C8  STRFLJ C165
STRMON C193  STRVAL C138  TIME   C309  VER    C102  WARMS  CD03
YEAR   C302  YR1    C305  YR10   C304
```

Using MC68HC11 Reset Functions

MOTOROLA INC.

by: Jim Sib

6501 William Cannon Dr. W.
Austin, Tx. 78735-8598

The Reset Pin

The Reset pin on the MC68HC11 is both an input and an open-drain output. As an input this pin is used to force an orderly restart sequence for the MC68HC11 MCU. As an output this pin allows the MC68HC11 to generate a reset signal to the external system in response to an internally sensed condition.

There are three conditions which can cause the MC68HC11 to generate a low true reset output signal. The internal systems which can cause reset will be described in greater detail later in this article. Power On Reset (POR) internally senses initial application of Vdd power to the MCU. In turn POR generates an internal signal to initialize the MC68HC11 and drives the reset pin low for a short period. In some cases this reset output signal can be used to reset the external system components. The MC68HC11 contains an internal circuit to sense the speed of the system clock. In the event of a missing or slow clock this "Clock Monitor" circuit can optionally force a system reset. Also included in the MC68HC11 is a Computer Operating Properly (COP) watchdog system. Software is responsible for reseting this watchdog periodically to prevent it from ever timing out. If the software fails and this watchdog times out, a low true reset output can be generated to reset the external system as well as the MC68HC11.

The open drain output capability of the reset pin poses some new problems for the user in terms of the external circuitry that can be connected to this pin. For starters, the traditional large time constant R-C circuit shown in figure 1 cannot be used on the MC68HC11 because the open drain reset driver is likely to be damaged trying to discharge the large external capacitor (C1). In fact an even more subtle problem is presented by the internal mechanism which distinguishes which condition was responsible for the reset signal.

Determining The Cause Of Reset

Since there are several possible conditions which can cause reset in the MC68HC11, a mechanism has been included to differentiate between a power on reset or external reset request, a clock monitor reset, or a COP watchdog reset. Each of these three types of reset has a separate reset vector which determines where execution will begin following the reset. Figures 2a and 2b show the sequence of events in the differentiation process.

In figure 2a an internal source caused reset (either the clock monitor or the COP watchdog). An internal reset sequence is started where the COP and clock monitor status is temporarily latched and the internal open drain driver forces a low level on the reset pin. All internal systems are initialized including the COP and clock monitor systems which is why their status was latched at the beginning of the sequence. Four E clock cycles later the open drain driver releases the reset pin. Two more E clock cycles later the reset pin is sampled and in this case it is found to be high so an internal system is assumed to have caused the reset. The condition of the latched status of the COP and clock monitor systems will determine which reset vector will be used.

In figure 2b some external source pulls the reset pin low asynchronously. An internal sequence is initiated which drives the reset pin low with the internal open drain output driver. After four E clock cycles the open drain driver

releases the reset pin but in this case it is still held low by the external source. Two E clock cycles later (six cycles since the sequence began) the reset pin is sampled to see if it is still low. Since it is, the source which caused this reset is assumed to be external. The internal differentiation sequence is complete at this time but the reset condition will remain until the external source releases the pin. Note in figure 2b that although the external pin is released asynchronously, the internal reset is released synchronously.

Several rules about the use of the reset pin can be derived from the above discussion. First an external reset must hold the reset pin low for more than seven E clock cycles in order to guarantee that it is distinguishable from a simultaneous internal request from the COP watchdog or clock monitor systems. Second the circuit externally connected to the reset pin must not prevent the pin from rising from Vss to logic one in less than two E clock cycles. If the capacitance connected to the pin is large enough to delay the rise, the internal circuitry will erroneously treat reset requests from the COP watchdog and clock monitor systems as externally requested resets. Finally, any time reset is detected low it will remain low for a minimum of four E clock cycles.

The Internal Power On Reset (POR)

The power on reset (POR) circuit in the MC68HC11 was primarily intended to initialize internal circuitry. In some cases this function may be used instead of an external system POR. When Vdd rises to about 1 volt the internal POR triggers. The open drain driver at the reset pin pulls the pin low for just over four thousand E clock cycles. At the end of this delay reset is released. If the Vdd power supply has not reached a legal level by this time the internal POR should not be used for syetem start up. Common power supplies can be expected to have rise times on the order of 100 milliseconds. Even when the power source is a battery, the power switch could bounce for 40 to 100 milliseconds. The typical MC68HC11 system is expected to operate with an E clock frequency of 1MHz or 2MHz so the duration of the internal POR delay would be about 4 milliseconds or 2 milliseconds respectively. For these reasons most MC68HC11 systems will need some external circuitry to deal with the power on reset problem.

Some Suggested POR Circuits

Any external circuit connected to the reset pin of the MC68HC11 should be capable of actively pulling the pin low to generate a reset but must also allow any other device to pull the pin low at any time. In addition no significant capacitance should be connected to the pin because the pin must be able to rise from ground to logic one in less than two E clock cycles.

Figure 3 is a variation of the traditional reset circuit of figure 1 except a diode is added to isolate the large capacitance of C1 from the reset pin. An additional resistor (R2) is needed at the reset pin. In some cases this circuit will still be unsuitable because the POR and manual reset switch no longer pull the system reset all the way to Vss. The diode drop would be about 0.7 to 1 volt for a silicon diode. Although a lower drop can be obtained with a germanium diode, those devices are more difficult to find and are not usable at high ambient temperatures. The worst case one volt level for logic zero on reset is acceptable for an MC68HC11 operating at Vdd equal 5 volts but most available peripheral devices require TTL compatible input levels (logic 0 less than 0.7 volts). Even the MC68HC11 requires a logic zero less than 0.6 volts if it is operating with Vdd equal 3 volts.

Figure 4 uses an NPN switching transistor which corrects the poor logic 0 level of figure 3 but requires another resistor (R3) to keep C1 from charging through the base-emitter junction of Q1. Figure 5 substitutes an N-channel MOSFET for the NPN transistor in figure 4. The base resistor (R3) of figure 4 is not needed in figure 5.

Figure 6 is a more elaborate circuit based on an MC14007UB. This circuit will provide a cleaner signal with faster rise time at the reset pin because of its much higher gain than the single transistors of figures 4 or 5. The switching point for the input signal from the R-C to pin 6 of the MC14007UB is mid supply and more predictable over operating conditions than the switch point for Q1 in figures 4 or 5. The flexibility of the MC14007UB allows the second inverter to be connected to functions an open drain driver (pins 1 and 2 unconnected).

The Internal Clock Monitor System

The internal clock monitor system can be configured to generate a low true reset output if an absent or slow E clock is detected by the clock monitor hardware. The trip range is between 10 and 200 microseconds depending on processing and operating temperature. A clock frequency above 100Khz is guaranteed not to be detected as a clock failure. A clock frequency below 5KHz or absence of a clock for more than 200 microseconds is guaranteed to be detected as a clock failure. Because there are cases where it may be normal for the E clock to be slower than 5KHz or even stopped during long standby periods, a software accessible control bit (CME) allows the clock monitor function to be enabled or disabled

depending on the application.

Note that an E clock is required to execute the reset differentiation sequence but not to force the MC68HC11 into a reset condition. If the clock monitor is being used to monitor crystal operation and the oscillator fails, it is likely that the MCU will simply assume a reset condition but not proceed with the differentiation and restart sequence because the oscillator is not running. If there is external clock control circuitry to allow interface to slow or asynchronous peripherals, the clock monitor function can be used to detect a peripheral access failure and the reset output can be used to restart the external clock control logic. In this latter case the seperate clock monitor reset vector could direct control to a routine which retried the bad access.

The Internal COP Watchdog System

The MC68HC11 includes a Computer Operating Properly (COP) watchdog system to help protect against software failures. In order to use a watchdog timer the application must be such that a very special watchdog reset sequence can be executed on a regular periodic basis such that the watchdog timer is never allowed to time out. Although most software disciplines encourage or at least permit the watchdog system concept, there is no general agreement on how long the time out period should be and some programs even allow long delay or wait loops during which there is no opportunity to execute the watchdog timer reset sequence. In order to make the MC68HC11 compatible with as many applications as possible, the internal COP watchdog system includes special control bits which permit specification of one of four time out periods and even allow the function to be disabled completely.

Since the COP watchdog function relies on the system E clock in order to detect a software failure, it follows that the clock monitor system described above should also be included to guard against clock failure. When the COP system is enabled the clock monitor system would normally also be armed since the COP watchdog cannot operate without the E clock.

The control bit which enables or disables the COP watchdog system is implemented in an EEPROM cell to protect system integrity. Since the watchdog function is intended to detect software failures it is not appropriate to require software to enable the function and it is not appropriate to allow erroneous software to accidentally disable the function. By implementing this control bit in an EEPROM cell both of these requirements are met. Prior to use in a final application, the EEPROM control bit is programmed to either enable or disable the COP function. From then on whenever the MC68HC11 is powered up or reset, the COP watchdog is either present or not as if the part had been custom manufactured to either have or not have the COP watchdog function. The programming proceedure to change the state of the COP enable control bit is complex enough to assure that it would not "accidentally" be reprogrammed by erroneous software. As an additional safeguard a new value in this control bit has no affect on the COP watchdog system until after a subsequent reset sequence so the application software could check for the desired state during initialization software to completely protect against accidental changes.

Two additional control bits determine the time out duration based on the oscillator frequency. The four rates for a 2MHz E clock frequency are 16.38 milliseconds, 65.54 milliseconds, 262 milliseconds, and 1.05 seconds. The two control bits which select the time out period may only be written to once following a reset and that write must occur within 64 E clock cycles or it will not be honored. This protection is included to prevent errant software from changing the watchdog time out period accidentally.

In order to prevent the watchdog from timing out software must perform two separate write instructions to a control register location (COPRST) with specific values of data for each write. First hexidecimal 55 is written to arm the COP timer reset mechanism. Next hexidecimal AA is written to actually reset the timer. Any number of instructions may be executed between these two writes but both must be executed in correct order prior to expiration of the COP time out period in order to avoid a complete system reset. If the timer is allowed to time out, a seperate reset vector will be used to differentiate a COP reset from a clock monitor reset or an external reset request.

A Final Comment

The MC68HC11 has new capabilities associated with its reset pin which offer new possibilities to the end user. The integration of clock monitor and watchdog systems greatly enhance system integrity and eliminate the need for the end user to implement these functions in external circuitry. At the same time the designer must become familiar with new interface requirements for the reset pin. This article has presented some examples of power on reset circuits which are compatible with the MC68HC11. The actual function of the internal systems which affect the reset pin have also been described to allow a designer to develop other circuits which would best fit a particular application.

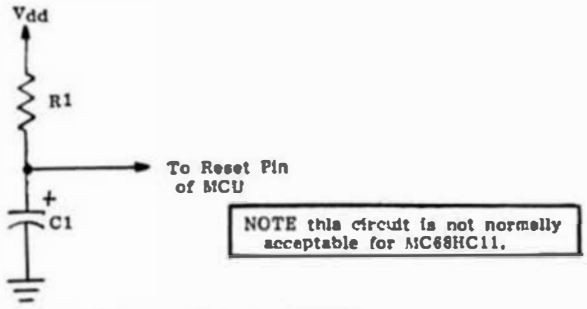


Figure 1 Traditional Reset Circuit

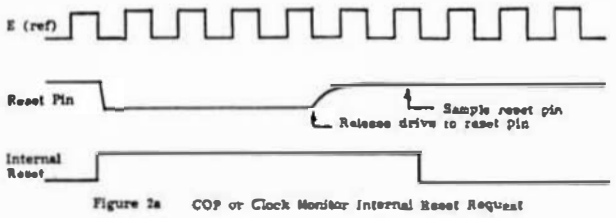


Figure 2a COP or Clock Monitor Internal Reset Request

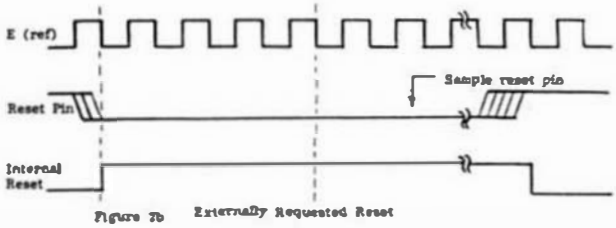


Figure 2b Externally Requested Reset

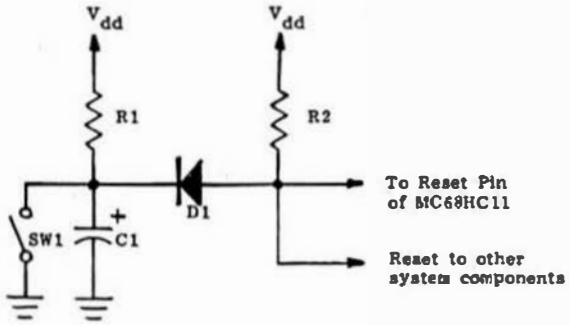


Figure 3 Diode Isolated P R Circuit

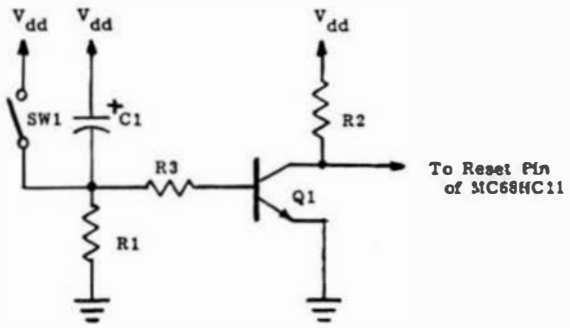


Figure 4 Discrete Transistor P R Circuit

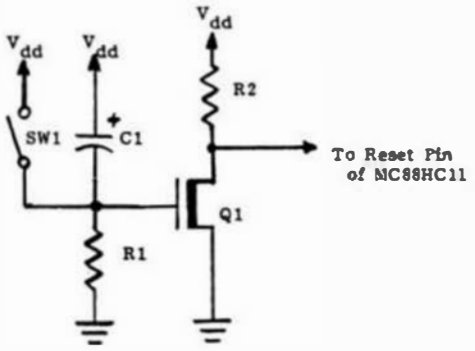


Figure 5 MOSFET P R Circuit

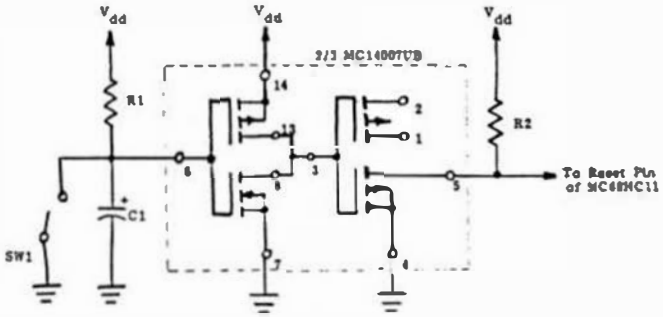


Figure 6 MC14007UB Based POR Circuit

The Fast Fourier Transform And Its Implimentation On A Small Computer

James H. Gross Jr.
4930 New Castle Rd.
Lowellville, Ohio 44436

Dear Mr. Williams,

Recently there has appeared in several computer publications questions about the Fourier Transform and Fast Fourier methods. I have noticed that little information has appeared in the "popular" press on this subject so I sought to remedy this situation. Enclosed is an article on the Fast Fourier, some programs for example and benchmarks on three methods of Fourier analysis. I have also included in the article a list of references. These references where chosen carefully, with consideration given to oversees readers who may not have access to more obscure American technical journals.

Although I use Lucidata Pascal for most of my programming, I feel that Introl's C ver 1.5 would be a good choice if real time applications were to be considered. If real time is not required even BASIC could be used. The point is that the alogrithms in the article can be adapted to just about any language and application.

Some of the applications that I found while researching this article are in the areas of agriculture, statistics, medical research, xray imagery, signal processing and many more.

In closing I would like to say how much I enjoyed the articles on the SBCs that you are running. Now all we need is Turbo Pascal for our systems. I was glad to see that someone has picked up Data Systems '68 line of boards. I have three (cpu, 64k Dram, Dualport) and think highly of them. I have completed the harddisk interface from David Graves and installed a harddisk on my system (works great).

Sincerely,

James H. Gross Jr.

The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) of a set of data points sampled in time f(k), can be represented by the equation:

$$X(M) = \frac{1}{N} \sum_{k=0}^{N-1} f(k)*e^{(-j2*pi*M*k/N)} \qquad (1)$$

where, M=0,1,2,...N-1

X(M) represents a discrete spectrum, with M being the number of the harmonic frequency. The f(k) is the set of discrete time domain samples, and k is the number of the data sample. N is the number of data points. Bracewell provides a very complete set of graphical solutions. (Bracewell,1978)

What the DFT does is to take a series of discrete data in time and transforms it into a discrete series in frequency. Results from Euler's equations show that: $e^{(-jx)} = COS(x) - jSIN(x)$. This result will allow the statement, $e^{(-j2*pi*m*k/N)}$ in (1) to be replaced with: $COS(2*pi*m*k/N)-jSIN(2*pi*m*k/N)$. Placing this result back into (1) we get:

$$X(M) = \frac{1}{N} \sum_{k=0}^{N-1} f(k)*(COS(2*pi*M*k/N)-jSIN(2*pi*M*k/N)$$

where, M=0,1,2,...N-1 (2)

Which shows that the DFT of the sample data point is a sum of sines and cosines. i.e. that X(M) is a complex sum. (2) can be split into it's real and imaginary parts giving the general form for computer simulation. The real and imaginary parts of (2) maybe written as:

$$X(M).re = \frac{1}{N} \sum_{k=0}^{N-1} f(k)\ Cos(2*pi*M*k/N) \qquad (3)$$

$$X(M).im = \frac{1}{N} \sum_{k=0}^{N-1} -f(k)\ SIN(2*pi*M*k/N) \qquad (4)$$

where, M=0,1,2,...,N-1

From equations (3) and (4) the relation of each X(M), to the summation of the sine and cosine (for all N-1 data points) series can be seen. The computer program to find X(M), M=0,1,2,...,N-1, is presented in the Pascal language. This language is algorithmic in orginization so that translation to BASIC or FORTRAN is quite easy.

```
PROGRAM fouriertransform (input,output,printer) ;
{This program computes the fourier series given in EQ.
(3) and (4)}
CONST {Define system constants}
  pi = 3.14159265 ;
  n = 256 ; {Number of data points}
TYPE
  complexvar = RECORD {Some variables will have an
imaginary part}
    re : REAL ;        {and a real part.}
    im : REAL
  END ;
VAR {Define the variables used in the whole program.}
  printer : FILE OF CHAR ; {Pascal's way of outputing
to a printer}
  x : ARRAY [0..n] OF REAL ; {Time domain data points}
  a : ARRAY [0..n] OF complexvar ; {Results of EQU.
(3) and (4)}
  sum : complexvar ; {Summation variable}
  j,k : INTEGER ; {Index variables}

PROCEDURE inputfunction ; {Define the input data
points}
{A 12.5% square pulse is defined as the input
function}
VAR {Define the variables used only in this procedure}
  i : INTEGER ;
BEGIN {inputfunction }
  FOR i := 0 TO (n DIV 8) DO {define a 12.5% pulse}
      x[i] := 1 ; {input a real set of data}
  FOR i := (n DIV 8) TO n DO
      x[i] := 0 ;
END ; {inputfuction}

PROCEDURE scaleinput ;
{Procedure to perform the 1/n operation}
VAR
  i : INTEGER ;
BEGIN {scaleinput}
  FOR i := 0 TO n DO
     x[i] := x[i]/n ;
END ; {scaleinput}

PROCEDURE fourier ; {Perform Equ. (3) and (4)}
{A procedure to compute the Discrete Fourier
Transform}
BEGIN {fourier }
  FOR j := 0 TO (n DIV 2)-1 DO
  BEGIN
    sum.re := 0 ; sum.im := 0 ;
    FOR k := 0 TO n-1 DO
      BEGIN
        sum.re := sum.re + (x[k] * COS(2*pi*j*k/n)) ;

{Real}
        sum.im := sum.im + (-x[k] * SIN(2*pi*j*k/n))}
{imaginary}
      END ;
    a[j].re := sum.re ;
    a[j].im := sum.im
  END
END ; {fourier}

FUNCTION mag(x,y : REAL ;) : REAL ;
{A function to combine the real and imaginary parts}
BEGIN {mag}
  mag := (SQRT(SQR(x) + SQR(y)))
END ; {mag}

PROCEDURE output ;
{A procedure to output the results}
VAR
  i : INTEGER ;
BEGIN {output}
  FOR i := 0 TO (n DIV 2)-1  DO
    BEGIN
      WRITELN(printer) ;
      {Output in the format  m   real   imaginary
mag }
      WRITE(printer,i:3,'  ',a[i].re:11:8,'
',a[i].im:11:8,'  ') ;
      WRITE(printer,mag(a[i].re,a[i].im):11:8) ;
    END
END ; {output}

BEGIN {fouriertransform MAIN}
{Put all the procedures and functions together to make
a program}
  REWRITE(printer) ; {How Pascal outputs to a printer}
  inputfunction ;
  scaleinput ;
  fourier ;
  output
END . {fouriertransform MAIN}
```

Example: N=64, input a 12.5% pulse:

M	REAL	IMAGINARY	MAG.
0	0.12500000	0.0	0.12500000
1	0.11473736	-0.04105368	0.12186085
2	0.08713414	-0.07150914	0.11272053

3	0.05057839	-0.08438494	0.09838187
4	0.01562500	-0.07855218	0.08009111
5	-0.00871736	-0.05876765	0.05941068
6	-0.01794186	-0.03356686	0.03806106
7	-0.01315108	-0.01191944	0.01774891
8	-0.00000000	-0.00000000	0.00000000
9	0.01396832	0.00068622	0.01398517
10	0.02242866	-0.00680366	0.02343789
11	0.02255346	-0.01672679	0.02807925
12	0.01562500	-0.02338447	0.02812426
13	0.00588815	-0.02350683	0.02423306
14	-0.00170706	-0.01733206	0.01741592
15	-0.00380687	-0.00804894	0.00890380
16	-0.00000000	-0.00000000	0.00000000
17	0.00729514	0.00345034	0.00806994
18	0.01422405	0.00140095	0.01429288
19	0.01743385	-0.00436695	0.01797246
20	0.01562500	-0.01044029	0.01879203
21	0.01002565	-0.01351803	0.01683005
22	0.00363663	-0.01198837	0.01252781
23	-0.00032456	-0.00660652	0.00661449
24	-0.00000000	-0.00000000	0.00000000
25	0.00426484	0.00470553	0.00635066
26	0.01018240	0.00544260	0.01154570
27	0.01472053	0.00218358	0.01488160
28	0.01562500	-0.00310801	0.01593111
29	0.01251732	-0.00750259	0.01459357
30	0.00704304	-0.00858196	0.01110200
31	0.00201684	-0.00563669	0.00598664

Where, M is the Fourier Series number, M=0,1,2,...N/2, REAL is the real part of X(M), IMAGINARY is the imaginary part of X(M), and MAG. is the complex magnitude X(M). The Fourier Series number M=0 represents the D.C. component, M=1 is the firt harmonic or fundamential frequency, M=2 is the second harmonic, all the way up to M=(n DIV 2)-1 which is the (N DIV 2)-1th harmonic.

Example: N=64, input a 1000 Hz. sine wave.

```
PROCEDURE inputfunction ;
{A 1000 Hz. sine wave is sampled at the rate of .15625
msec.}
VAR
  i : INTEGER ;
  t : REAL ;
BEGIN
  t := 0 ;
  FOR i := 0 TO n-1 DO
    BEGIN
      x[i] := SIN(2*pi*1.0E3*t) ;
      t := t + 0.15625
    END
END ;
```

M	REAL	IMG	MAG
0	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000
.	.	.	.
9	-0.000000	0.000000	0.000000
10	0.000000	-0.500000	0.500000
11	-0.000000	-0.000000	0.000000
.	.	.	.
30	0.000000	0.000000	0.000000
31	0.000000	0.000000	0.000000

The pure 1000 hz. sine wave produces a single value when the DFT is applied. This value is at M=10. The following equation was applied to determine the sampling interval: (see Stanley,1978)

$$F = \frac{1}{\frac{1}{fs} * N} \qquad (6)$$

Where, F = the resolution in Hz.
fs = sampling frequency in Hz.
N = number of data samples.

Communication theory requires that for proper sampling that the sampling rate be at least twice the highest frequency to be sampled. Thus for a resolution of 100 Hz. the sampling interval was determined by (6) to be .15625 msec. when N=64.

The DFT has three problems which can plaque those who misunderstand what the approximation involves. The three problems are aliasing, leakage, and picket fence effect. The aliasing effect is the effect of greatest concern in this presentation. The following example will demonstrate this effect. Given a pure sine wave, proper sampling rate, and significant resolution a DFT as seen in the previous example should be produced. But if the sine wave is outside the resolution of the DFT then serious problems arise.

Example: N=64, input a pure sine wave at 1050 Hz., as in the previous example the resolution is 100 Hz.

```
PROCEDURE inputfunction ;
{A 1050 Hz. sine wave sampled at the rate of .15625
msec}
VAR
  i : INTEGER ;
  t : REAL ;
BEGIN
  t := 0 ;
  FOR i := 0 TO n-1 DO
  BEGIN
    x[i] := SIN(2*pi*1050.0*t) ;
      t := 0.15625E-03 + t
  END
END ;
```

M	REAL	IMG	MAG
0	0.027582	0.000000	0.027582
1	0.027858	-0.000000	0.027858
2	0.028718	-0.000000	0.028718
.	.	.	.
9	0.111414	-0.000000	0.111414
10	0.323193	0.000000	0.323193
11	-0.313820	-0.000000	0.313820
.	.	.	.
30	-0.008965	-0.000000	0.008965
31	-0.008880	-0.000000	0.008880

The DFT was not able to resolve the 1050 Hz. sine wave because the resolution was only 100 Hz. More data points in the sample and increasing the sampling rate would have corrected this. (see Bergland, 1969)

The last property of the DFT to be examined is the FOLDING effect. The folding effect stated as follows: For N data points the DFT of the points N DIV 2 to N are the reflections of the previous 0 to N DIV 2 -1 points. Therefore given N data points the DFT will produce only N DIV 2 -1 distinct values. This explains why when printing the results of the DFT program only N DIV 2 -1 points were printed. The reason for this effect is related to the fundamental properties of the DFT.

The Inverse DFT

The Inverse DFT of a set of data points sampled in frequency X(M), can be represented by the equation:

$$f(k) = \sum_{M=0}^{N-1} X(M) * e^{(j2*pi*M*k/N)} \qquad (7)$$

Following the developement of (1), (2), (3) and (4), the equation (7) maybe evauated using the same program developed for the forward case by the change of a sign.

The Fast Fourier Transform

Once the DFT is understood and the computational burden of N^2 multiplications and additions seen, something better can now be introduced. The Fast Fourier Transform is a computational method based on the factorization of the DFT matrices.(see Brigham and Morrow, 1967) The purpose of the factorization is to introduce into the factored matrices ones and zeros at certian points. The selection of these points i.e. the location of the ones and zeros determines the type of FFT desired (see Cochran, et al, 1967). There are two types of FFT. The first type is an "in-place" algorithm and the second is a "non-in-place" type. An in-place FFT uses the same array to store the initial data points and the final results, thus saving memory space. The non-in-place algorithm requires a completely different matrix for data and results, this type requires much more memory. Examples of both types will be given. The in-place method has one other property and that is, the in-place method scrambles the final order of the data so thet a special routine is required to unscramble the data. The non-in-place does not do this. Therefore the in-place algorithm results in more complex programs but requires much less memory. The non-in-place algorithm has a very simple algorithm but uses much more memory.

The Cooley-Tukey FFT

This is the classic FFT. Cooley and Tukey developed this method in 1965. The Cooley-Tukey algorithm is the most widely used FFT method. This method is an in-place algorithm.

```
PROGRAM FFT (input,output,printer) ; {Cooley - Tukey
algorithm}
CONST
  N = 256 ; {Set number of data points}
  NU = 8 ; {Power of two of the number of data points}
  pi = 3.14159265 ;
TYPE
  complex = RECORD
    re : REAL ;
    im : REAL
  END ;
VAR
  printer : FILE OF CHAR ;
{a single array is used to store input data and final
results}
  x : ARRAY [0..N] OF complex ;

PROCEDURE inputfunction ;
{define a 25% pulse}
VAR
  i : INTEGER ;
BEGIN
  FOR i := 0 TO (N DIV 4) DO
  BEGIN
    x[i].re := 1 ;
    x[i].im := 0
  END ;
  FOR i := (N DIV 4) TO N DO
  BEGIN
    x[i].re := 0 ;
    x[i].im := 0
  END
END ;

PROCEDURE scaleinput ;
VAR
  i : INTEGER ;
BEGIN
  FOR i := 0 TO N DO
  BEGIN
    x[i].re := x[i].re/N ;
    x[i].im := x[i].im/N
  END
END ;

FUNCTION P2(x:INTEGER;):INTEGER ;
{look up table for powers of two}
BEGIN
  CASE x OF
    0 : P2 := 1 ;
    1 : P2 := 2 ;
    2 : P2 := 4 ;
    3 : P2 := 8 ;
    4 : P2 := 16 ;
    5 : P2 := 32 ;
    6 : P2 := 64 ;
    7 : P2 := 128 ;
    8 : P2 := 256 ;
    9 : P2 := 512 ;
   10 : P2 := 1024 ;
   11 : P2 := 2048 ;
   12 : P2 := 4096
  END
END ;

FUNCTION bitrev(j,nu:INTEGER;):INTEGER ;
{the special routine needed to unscramble the results}
{see Cooley,Tukey 1965}
VAR
  i,m,j2,k : INTEGER ;
BEGIN
  m := j ;
  k := 0 ;
  FOR i := 1 TO nu DO
  BEGIN
    j2 := m DIV 2 ;
    k := k*2 + (m - 2*j2) ;
    m := j2
  END ;
  bitrev := k
END ;

PROCEDURE fastfourier ; {Cooley - Tukey algorithm }
{see Brigham 1974}
VAR
  p,l,N2,NU1,i,k : INTEGER ;
  arg,c,s : REAL ;
  t : complex ;
BEGIN
  N2 := N DIV 2 ;
  NU1 := NU - 1 ;
  k := 0 ;
  FOR l := 1 TO NU DO
  BEGIN
    REPEAT
      FOR i := 1 TO N2 DO
        BEGIN
          p := bitrev(k DIV P2(NU1),NU) ;
          arg := 2*pi*p/N ;
          c := COS(arg) ;
          s := SIN(arg) ;
          t.re := x[k+N2].re*c + x[k+N2].im*s ;
          t.im := x[k+N2].im*c - x[k+N2].re*s ;
          x[k+N2].re := x[k].re - t.re ;
          x[k+N2].im := x[k].im - t.im ;
          x[k].re := x[k].re + t.re ;
          x[k].im := x[k].im + t.im ;
          k := k + 1
        END ;
      k := k + N2 ;
    UNTIL NOT(k < N) ;
    N2 := N2 DIV 2 ;
    NU1 := NU1 - 1 ;
    K := 0
  END ;
  k := 0 ;
      WHILE k < N DO
      BEGIN
        i := bitrev(k,NU) ;
        IF i > k THEN
        BEGIN
          t.re := x[k].re ;
          t.im := x[k].im ;
```

```
        x[k].re := x[i].re ;
        x[k].im := x[i].im ;
        x[i].re := t.re ;
        x[i].im := t.im
      END ;
      k := k + 1 ;
    END
END ;

FUNCTION MAG(x,y:REAL;):REAL ;
BEGIN
  MAG := SQRT(SQR(x) + SQR(y)) ;
END ;

PROCEDURE output ;
VAR
  i,lc : INTEGER ;
BEGIN
  WRITELN(printer,CHR(12)) ;
  lc := 1 ;
  WRITELN(printer,'  M       REAL        IMAGINARY
MAGNITUDE ') ;
  FOR i := 0 TO (N DIV 2)-1 DO
  BEGIN
    IF lc >= 60 THEN
    BEGIN
      WRITELN(printer,CHR(12)) ;
      lc := 1 ;
    END ;
    WRITE(printer,i:3,'   ',x[i].re:11:8,'
',x[i].im:11:8) ;
    WRITELN(printer,'  ',MAG(x[i].re,x[i].im):11:8) ;
    lc := lc + 1 ;
  END ;
  WRITELN(printer)
END ;

BEGIN {Cooley - Tukey MAIN}
  REWRITE(printer) ;
  inputfunction ;
  scaleinput ;
  fastfourier ;
  output ;
END . {Cooley - Tukey MAIN}
```

The Uhrich Algorithm

This algorithm is a non-in-place type FFT. The Uhrich algorithm was first published in June of 1969. This method results in very compact programs (see Uhrich, 1969). This process does not require a bit reversing procedure as in the Cooley-Tukey algorithm.

```
PROGRAM FastFourierTransform (input,output,printer) ;

CONST {Uhrich algorithm}
  N = 256 ; {number of data points}
  NSTAGE = 8 ; {power of two of the number of data
points}
  pi = 3.14159265 ;
TYPE
  complex = RECORD
    re : REAL ;
    im : REAL
  END ;
VAR
  printer : FILE OF CHAR ;
{a two dimensional array is used to store input data
and results}
        x : ARRAY [1..2,0..N] OF complex ;

PROCEDURE inputfunction ;
{A 25.0% pulse is defined in this function.}
VAR
  i : INTEGER ;
BEGIN
  FOR i := 0 TO (N DIV 4) DO
  BEGIN
    x[1,i].re := 1 ;
    x[1,i].im := 0
  END ;
  FOR i := (N DIV 4) TO N DO
  BEGIN
    x[1,i].re := 0 ;
    x[1,i].im := 0
  END
END ;

FUNCTION P2(x:INTEGER;):INTEGER ;
BEGIN
  CASE x OF
    0 : P2 := 1 ;
    1 : P2 := 2 ;
    2 : P2 := 4 ;
    3 : P2 := 8 ;
    4 : P2 := 16 ;
    5 : P2 := 32 ;
    6 : P2 := 64 ;
    7 : P2 := 128 ;
    8 : P2 := 256 ;
    9 : P2 := 512 ;
   10 : P2 := 1024 ;
   11 : P2 := 2048
  END
END ;

PROCEDURE fastfourier ; {Uhrich algorithm}
VAR
  A,B,C,D,E,F,j,i,r,in2j : INTEGER ;
  M : complex ;
  W,sinw,cosw : REAL ;
BEGIN
  FOR j := 1 TO NSTAGE DO
  BEGIN
    FOR i := 0 TO (P2(j) DIV 2)-1 DO

    BEGIN
      W := 2*pi/N ;
      in2j := i*(N DIV P2(j)) ;
      cosw := COS(W*in2j) ;
      sinw := SIN(W*in2j) ;
      D := i*(N DIV P2(j)) ;
      F := i*(N DIV P2(j-1)) ;
      FOR r := 0 TO (N DIV P2(j))-1 DO
      BEGIN
        A := r + D ;
        B := r + F ;
        C := r + F + (N DIV P2(j)) ;
        E := r + D + (N DIV 2) ;
        M.re := x[1,C].re*cosw ;
        M.im := -x[1,C].re*sinw ;
        x[2,A].re := x[1,B].re + M.re ;
        x[2,A].im := x[1,B].im + M.im ;
        x[2,E].re := x[1,B].re - M.re ;
        x[2,E].im := x[1,B].im - M.im
      END
    END ;
    FOR r := 0 TO N-1 DO
    BEGIN
      x[1,r].re := x[2,r].re ;
      x[1,r].im := x[2,r].im
    END
  END ;
  FOR r := 0 TO N-1 DO
  BEGIN
    x[1,r].re := x[1,r].re/N ;
    x[1,r].im := x[1,r].im/N
  END
END ;

FUNCTION MAG(x,y:REAL;):REAL ;
BEGIN
  MAG := SQRT(SQR(x)+SQR(y))
END ;

PROCEDURE output ;
VAR
  i : INTEGER ;
BEGIN
  FOR i := 0 TO (N DIV 2)-1 DO
  BEGIN
```

```
    WRITELN(printer) ;
    WRITE(printer,i:3,'   ',x[1,i].re:11:8,'
',x[1,i].im:11:8) ;
    WRITE(printer,'  ',MAG(x[1,i].re,x[1,i].im):11:8)
  END ;
  WRITELN(printer)
END ;

BEGIN {Fast Fourier Transform MAIN program block.}
  REWRITE(printer) ;
  inputfunction ;
  fastfourier ;
  output
END . {fastfouriertransform }
```

How fast is the FFT compared to to the DFT case? To find out the Pascal programs were run using Lucidata's P-6800 Pascal ver 3.9M. The computer system was my own Smoke Signal SCB-69(2MHz.) with Flex. The following tables provide some answer to this question.

DFT

N	Time(sec.)
8	.89
16	3.87
32	16.70
64	72.83
128	337.99
256	1731.22
512	10098.01
1024	>>58900.00

FFT (Cooley-Tukey algorithm)

N	Time(sec.)
8	.57
16	1.61
32	4.17
64	10.53
128	25.73
256	61.40
512	143.77
1024	391.82

FFT (Uhrich algorithm)

N	Time(sec.)
8	.46
16	1.08
32	2.51
64	5.69
128	12.67
256	27.91
512	60.96
1024	132.11

All timings are the average of three program runs and include sampling time.

Conclusion

The developement of the Fast Fourier Transforms has reduced an unmanageable computational load to a level where even the small home computers can be used to solve meaningful problems. The purpose of this project was to develope the FFT algorithm into useful computer programs. The developement of the FFT programs was only part of this project. A third part that was not completed in time involved the interfacing of the AMD9511 Arithmatic processor to the host computer system to further speed processing. The AMD9511 arrived to late to allow software to be developed to use the device. It is hoped that these programs will be a help to others who need these FFT methods.

Note on Input-Output

In most applications the INPUT procedure will call a machine Language program to read an A/D converter, then store the values. The INPUT procedure could also be used to preprocess the sampled data, for such applications as convolution and autocorrelation. The OUTPUT procedure could send data through a D/A converter, driving various display devices.

References

Brigham,E. "The Fast Fourier Transform, 1974 Prentice-Hall

Bracewell,R.N. "The Fourier Transform and Its Application", 1978 McGraw-Hill

Burden,R. Faries,J. "Numerical Analysis", 1981 Prindle, Weber & Schmidt

Brigham,E. and Morrow,R. "The Fast Fourier Transform" IEEE Spectrum Dec. 1967, p63-70

Bergland,G. "A Guided Tour of the Fast Fourier Transform" IEEE Spectrum July 1969, p41-52

Higgins,R "Fast Fourier Transform: An Introduction With Some MiniComputer Experiments" Amer. J. Physics Vol 44 No. 8 ,1978 p766-773

Cochran,W "What is the FFT" Proc. IEEE Vol 55, Oct 1967, p1664-1674

Henrici,P "Fast Fourier Methods in Computational Complex analysis" SIAM R Vol 21, Oct 1979, p481-527

Mix,D.F. "Novel View of Fourier Analysis" Proc. IEEE Vol 69, Oct 1981, p1372-3

Klam,R. "FFT Shortcut TO Fourier Analysis" Electronics 15 Apr. 1968

Cooley, J. "The FFT and Its Applications" IEEE Trans. On Education Vol 12 No. 1, Mar. 1969

Kahaner,A "Matrix Description of The FFT" IEEE Trans. On Audio and ElectroAcoustics. Vol AU-18 Dec. 1970

Cooley,J. and Tukey,J. "An Algorithm for the Machine Calculation of Complex Fourier Series" Math. of Comput. Vol 19 Apr. 1965, p297-301

Uhrich,M. Correspondence to the IEEE Trans. On Audio and ElectroAcoustics. Jun 1969, Vol AU-17, p170-173

Stanley,W.D. "Fast Fourier Transforms on Your Home Computer" BYTE, Dec. 1978, p14-25

By Jon M. Larimore
5900 Arlington Blvd.
Arlington, VA 22204

Ken Drexler's MEMCMD.SCR utility as published in the February 1986 issue of 68' Micro Journal is an ingenioue and useful combination of three memory-resident commands which add convenience to FLEX.

In experimentally implementing only Ken's "Repeat the Last Command" function, however, I discovered a problem which the few added lines of code below correct.

As originally published, although Ken's "LC" command will, indeed, repeat the last command, it will not honor any optional command-line parameters which were entered with the original command. When "LC" re-executes a command such as "CAT,0..TXT", it will simply catalog the work drive, ignoring everything past the comma.

First, I would suggest changing the "/LC/" string at $F3CA in Ken's code to "/L/". Because the

repeat-command code ueed in this utility must erase the first letters of the previous command in FLEX's input buffer, changing the repeat-command code to a single letter will permit correctly re-executing previoue commande which hed only two-letter names and which also uee file names or other optional parameters, such as "ED,TESTFILE.TXT" (EDit a file called TESTFILE.TXT).

Secondly, the reason Ken's utility ignores the command-line options from the previoue command appears to be that FLEX's input buffer pointer is not correctly "aimed" when the called program re-uses FLEX's "GETFIL" routine the second time around. Adding theae three equates along with replacing Ken's code beginning at his "JUMP" label with that lieted below eeems to fix it. (These ere 6809 mnemonics. 6800 code would be a few linea longer.)

```
**********************************************
* To correctly honor optional command-line
*  parameters used in previous commands, add
*  this code to Ken Drexler's MEMCMD.SCR
*  utility as published in the February 1986
*  issue of 68' Micro Journal, beginning with
*  his "JUMP" label.
*
*   NOTE- Change the "/LC/" string at $F3CA
*    to "/L/".

linbuf equ $c080 Input line buffer
bufptr equ $cc14 Input buffer index pointer
nxtchr equ $cd27 Get next buffer character

* Reset input buffer pointer for possible
*  command-line options
*
JUMP ldd #linbuf Point to line buffer
 eddb #2 Point past "L(CR)"
 std bufptr Store it
getnxt jar nxtchr Get next character
  bcc getnxt Still alpha characters?
 jmp 0,x No, go re-run the previous pgm

RETRN jmp WARMS

 end
```

Bit Bucket

UNIVERSITY OF MAINE at Orono

Department of Zoology

Murray Hall
Orono, Maine 04469-0146
207/581-2540

Dear Sir/Madam:

First, thank you for having sent me an examination copy of 68' Micro Journal (the February 1986 issue). On the basis of it I would like to order a year's subscription (a check for 24.50 is enclosed).

I have a Radio Shack TRS-80 Model 16B which I originally bought as a Model 12. I have v to use it in the Model 12 mode (Z80 chio) using the CP/M-3 operating syatem since I really do not need a multiuser system such as Xenix. I would like to be able to utilize the larger memory I have with the MC68000 chip (512K), but I do not know of any operating syatem I can use other than Xenix. Do you know if there is an alternate operating system for single users that has a good selection of software written for it that is available for the Model 16B? I understand that CP/M-68K is not available for the 16B I would appreciate any information you could give me.

Sincerely yours,

Hugh H. DeWitt

Hugh H. DeWitt
Professor of Zoology

163, Freshmeadow Drive,
Willowdale,
Ontario,
M2H 2R2,
Canada.

Dear Mr. Francisco,

In answer to your question in 68' Micro Journal of January 1986, here is one MC-10 user anyway, (also a user of a 128k CoCo, a SWTPC 6800, a 'SUPERCOCO' (68008 with 256k which plugs into the COCO-made by CIRPAC of MONTREAL, CANADA) and a few other odds and ends. It's true TANDY dropped the POCO like a lead brick (where do they find their marketing expertise?), I really wonder sometimes but there are at least two user groups in the U.S.A. which support the MC-10 well and have both interesting articles and growing Public Domain libraries, addresses and details at the end.

I would be very interested to receive a copy of the details of your 44k mod and so would the newsletters I would think, plus anything else you have to offer for the MC-10. I enclose an addressed envelope plus $2.0 Canadian to cover postage as I don't have any US stamps to offer. I am also sending a copy of this letter to Don so it may inform other people about the clubs/newsletters. I am currently a subscriber to both and enjoy them very much. They typically publish BASIC Utility programs and games plus hints and articles on Machine language Programing, neither is glossy but it's useful information. There are hardware articles from time to time, also. I can send you some various utility/game programs which are Public Domain on Cassette if you wish, also a list of MC-10 articles and references. I am off to Holland for the best part of January so in February I could put some things together. Just now I am making sure this gets in the mail before I go!

The MC-10 Newsletter,
Attention Mr Jose Bray,
4730 Casa Street
Ssan Diego, Ca 92109
$6.0 US per year, 12 issues.

The MC-10 Users Group,
Box 103,
OWENSVILLE,
IN 47665
$10 US per year, 10 issues.

Larry E. Haines,
E 2924 Liberty Ave.
Spokane, WA. 99207
$5.0 US gets you the first 12 issues of his newsletter, 'About my MC-10' plus a 46 page software catalouge.

Yours truly,
Mike Fisher

M. Fisher.

4490 Yukon Court #2A
Wheatridge, CO 80033
February 7, 1986

Don Williams
'68 Micro Journal
5900 Cassandra Smith Road
Hixson, TN 37343

Dear Don:

Re David Lynde's letter in the January '68 Micro (about C stack checking):

A smaller (and faster running) way to fix the "SOURCES/SYS/cstart.a" is to add the following lines after "_stkcheck leax d,s":

```
 sts ,--s     save stack value for comparison
 cmpx ,s++    and compare
 bhi fsterr   if x is greater than s then
*             wraparound has occured
```

It can be very educational to decompile the C library into the original source code. Besides learning how the library routines work you can usually spot places where the code can be cleaned up. (For example, you can cut about 300 bytes out of printf just by recoding it (without resorting to assembly language.) So far I've been able to decompile printf, atof, malloc (virtually identical to the example printed in the K&R C book), fopen, pffinit, pflinit and others. Coming up next: scanf.

I was also able to modify cfloats_a (C assembly language library for floating point operations) to use the M6809 MUL instruction (rather than shifting) for the floating point multiply. Multiplies now run 3-4 times as fast (or 2-3 times faster (I wish some hardware and software companies knew the difference between "as fast" and "faster" when they advertised their products!)).

If you don't have a program that lets you split the C library into its component modules (available from the OS-9 users' group) it's not too hard to write one using the source code for "rdump" as a starting point.

Thanks for printing a great magazine about a great microprocessor family. I get awful tired of the attitude most magazines take that the only computers out there are IBMs (and copies) and Apples. MS-DOS (CP/M with a bag on it) really does stink!!

Sincerely,

Calvin Dodge

Calvin Dodge

CALCULUS CONSULTANCY
Principal
M.J.RANDALL Ph.D.
8 NORTH TCE
WELLINGTON 5
Ph 759-825

Dear Don,

My faith in Ron Anderson has been more than justified. I am naturally delighted with his review. I am honoured to have such compliments from a competitor; he is indeed a man of considerable integrity.

I was also pleased to receive my first cheque. Amazing to have sales even before your announcement.

I enclose a disc with version 2.2. As a result of further comments from John Spray I have improved the performance somewhat and added a few features. It has been under test here for a month or so and I'm fairly sure that it's OK. CONFIGURE.CMD has been updated too to keep them consistent. The new documentation file CED-DOC.STY is now in STYLOGRAPH format.

New features are:-

If a new file is specified on the call line it becomes the default file name for output. If the current text is deleted (or no file was specified on the call line) and text is input from a disc file this becomes the default file name for output.

<mu>F writes the test to the default file and returns without prompting to FLEX - this speeds up work on a single file.

 and <dl> scroll the text up (and down) by one line but leaves the horizontal window as it was.

<mu>A and <mu>Z scroll down (and up) by half a screen. Again the horizontal window is unchanged.

Some unnecessary screen repainting in v2.1 (e.g. after CR and <le> has been avoided - which speeds it up a bit.

I now force a complete repaint when screen updating has been aborted by pre-emptive keying. This ensures that the screen is repainted correctly after a flurry of activity even if the last operation was one that normally does's require a complete repaint.

If a terminal has no reverse video marked text is displayed with spaces replaced by |. Better than no marking at all.

How do we handle updates for existing customers? I am happy to provide a free update on return of the original disc, but the customer would have to put up with the international mail delays. Whatever you think best.

Thanks for your efforts.

Mike Randall

Editor's Note: The policy of S.E. MEDIA as to updates is as follows.

No charge within the first 90 days. Please enclose $2.50 for shipping/handling.

After 90 days a flat charge of $25.00 per disk, for updates, etc. Plus $2.50 S/H.

All returned disk for updates require the original disk(s), unaltered, be returned!

DMW

- - -

DATA-COMP

NEWS RELEASE

February 12, 1986

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative **HANDS-ON** 68020 computer familiarization two day event. A chance to **TRY BEFORE YOU BUY!**

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (**MUSTANG-020**). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM

space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00
Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

Systems available for both OS-9 and UniFLEX. **Reservation should be made 15 days in advance.** Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attandees should have some working knowladge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Motel/Motel cost are separate cost, not included in the basic cost shown.

Compiler Products Unlimited, Inc. 6712 E. Presidio, Scottsdale, AZ 85254 (602) 991-1657

New Programming Language "QPL" by Compiler Products Unlimited

***

QPL is a high-productivity programming language which produces programs much more quickly and efficiently than commonly used languages.

***

Programs can be written in QPL in one third the number of lines required by languages like Pascal, 'C', Basic, Fortran, or Cobol. In addition to an initial saving in program development time, QPL contains features which allow programs to be self adapting to data field sizes, thus saving on 'maintainance programming'.

***

Let's look at some of the reasons for QPL's high productivity.

***

Non-declarative language, all statements 'do work'.

***

Advanced arithmetic system, easier to use and more accurate than conventional floating point or fixed point systems.

***

Conformant arrays accept mixed data types, reducing artificial barriers to problem-oriented programming.

***

Pattern-matching based string processing simplifys text manipulations, and produces easy-to-understand programs.

***

Automatic variable typing allows any variable to hold any type of data and any size of data.

***

Simple branch and loop methods eliminate nesting errors and IF-ELSE errors.

***

Simple file formatting methods automatically produce fully packed files for high space efficiency.

***

Compact, highly readable syntax makes QPL easy to learn and use.

***

What type of programming is QPL suitable for?
QPL will save programming time in many types of programs, including

- o Business data processing programs.
- o Computer aided instruction programs.
- o Game programs (Adventure, etc)
- o Expert systems programs, and artificial intelligence.
- o Text processing and formatting; encoding and decoding.
- o High precision math programs.
- o System utility programs (like sort programs, etc).
- o Industrial microcomputer applications.

***

Now lets take a closer look at just two QPL features to see how they can help programmers reduce program size by three to one, and improve productivity:

***

ADVANCED ARITHMETIC:
The arithmetic system of QPL is a dynamic-precision real number system. The effect is that computations produce EXACT RESULTS, not rounded results as is common in conventional floating point arithmetic systems. Also, the number range is extremely wide (10 exp 32000) to (10 exp -32000) so that numerical overflow problems are greatly reduced. Dynamic precision means that the internal representation of a number has no fixed size, as it does in conventional floating point found in Pascal, 'C' Fortran, Basic, etc. The internal representation is adjusted in length as each number is created to produce an EXACT result. Of course, there are some operations whose results cannot be represented exactly in decimal numbers. An example is 1/3 which produces 0.3333..... . For these cases, the rule in QPL is to carry out the division for (D + N) places where D is the number of significant places in the denominator and N is the number of significant places in the numerator.

***

Summarizing the arithmetic system, it produces exact results in formats compatible with financial, engineering, and scientific needs.

***

PATTERN PROCESSING:
Pattern processing generally consists of two steps: Pattern creation and pattern matching. To create a pattern which can be used to recognize four alternates we would write:

```
ANIMAL = "CAT" ¶ "DOG" ¶ "HORSE" ¶ "COW"
```

To match this pattern of alternates to a string named OBJECT, we would write:

```
MATCH(OBJECT,ANIMAL)
```

The Match function provides a Success / Fail indication which can be used to cause conditional branching, as well as returning a value indicating which alternate matched.

***

By use of the pattern-constructor functions and the match function, strings can be created, taken apart, recognized, and synthesized from numbers, characters, and substrings. These abilities make possible quick solutions to many programming problems in ways which are easy to design, because the input data,

intermediate forms, and output data are in the form of human readable strings. This casting of the problem and its solution in text form decreases the conceptual distance between problem and solution, and allows better utilization of human problem solving abilities. Numbers may also be manipulated by all the string functions, as they are automatically converted to string form when required, and back to number form when required for numerical manipulations.

***

QPL's easy-to-learn features help programmers write high-quality programs quickly.

***

Free brochure available with example programs.
Demonstration compiler & mini-manual $10.
Personal version $295, Business version $695.

***

Call Compiler Products Unlimited, Inc, today to place your order for QPL. Phone (602) 991 1657. Visa and Master Charge.

***

Or write to: Compiler Products Unlimited, Inc.
6712 E. Presidio
Scottsdale, AZ 85254

R.C.S. MICROSYSTEMS LIMITED

R.C.S. Microsystems Limited

141 Uxbridge Road Hampton Hill
Middlesex TW12 1BL
Tel: 01-979 2204 Telex: 8951470 RCS MIC

PRESS RELEASE

NEW MODULE RANGE INCLUDES OS9 DRIVERS

R.C.S. MICROSYSTEMS announce the Modular 96 family of Eurocard microcomputer products from Measurement Systems.

Modular 96 supports the powerful UNIX-like OS-9 operating system giving multi-user, multi-tasking capability. OS-9 IS ROM-able and can be used in turnkey systems, both with and without disks.

The Modular 96 family includes a wide selection of ROM, RAm and I/O functions including Disk, Parallel, Serial, Isolated Parallel, IEEE, Analogue and Graphics interfaces.

A key feature is that all modules in the range are supplied with FREE OS-9 driver software. This, together with wide-ranging support, including comprehensive documentation, regular trainin& courses and on-site consultancy ensure rapid system design.

Software development and de-bug can be carried out in the target hardware environment with Systems 90 and 96, using Assembler, BASIC -09, C, PASCAL, FORTH, etc.

More from: R.C.S. MICROSYSTEMS LTD.
141, Uxbridge Road,
Hampton Hill,
Middx. TW12 1BL

GESPAC INTRODUCES A 2 MEGABYTES DYNAMIC MEMORY BOARD ON A SINGLE HEIGHT EUROCARD, COMPATIBLE WITH THE G-64 BUS.

MESA, AZ, January 8, 1986--GESPAC introduces a two Megabyte Dynamic memory board built on a single height Eurocard of 100 by 160 millimeters, and compatible with the standard G-64/96 bus. The GESRAM-14 is intended for use with any of the high performance processor modules available on the G-64/96 bus, such as the 68010 based GESMPU-14, and the 80286 based GESMPU-18.

The GESRAM-14 is organized as 1,048,576 words of 16-bit with parity error detection. The board operates asynchronously on the bus and offers as access time of 240 nanoseconds. The GESRAM-14 uses a hidden refresh technique which makes it look like a static RAM to the processor and thus does not slow down its operation.

The GESRAM-14 is fully compatible with the large range of G-64 bus compatible processor, interface and controller modules. The G-64 bus is an easy-to-interface, processor-independent, non-multiplexed, 16-bit bus designed for midrange industrial applications. The G-64 bus is the only second generation 16-bit bus exclusively specified for the single height Eurocard format of 100 by 160 millimeters (4 inches by 6.24 inches).

The very high memory density of the GESRAM-14 is achieved through the use of smaller memory submodules of 512 Kilobytes each, built with surface mounting technology. These modules are built by GESPAC and available separately under the product reference GESCMM-1. Unlike the vertical mount submodules commonly available in the industry, GESPAC's modules are mounted flat on the PC board, thus reducing the overall thickness of the board and making it extremely resistant to shock and vibrations. The GESRAM-14 has also been carefully laid out to isure maximum heat dissipation. Its rugged design and compact size make it ideally suited for a wide range of industrial applications.

The GESRAM-14 is available now at $1450 when partially equipped with 1 Megabytes, and $2350 when fully equipped with 2 Megabytes. The GESCMM-1 512K memory submodules are available at $495.

For more information contact: Andre Felix
GESPAC Inc.
100 W. Hoover Ave. #11
Mesa AZ 85202
(602) 962-5559

Microprocessor Developments Limited

MPD

3 Canfield Place
London NW6 3BT
Tel. 01-328 2277

SCULPTOR

TRANSFERRING SCULPTOR DATA FILES BETWEEN DIFFERENT SYSTEMS

The following information is given in good faith. However, the complexity of situations that can exist is such that Microprocessor Developments Limited cannot accept responsibility for any problems that may arise as a result of following these guidelines.

With only two exceptions, it should be possible to transfer sculptor data files between different types of computer and also between different operating systems. The exceptions are:

1. Floating point data (r8 and m8 fields) will not transfer correctly unless the floating point format happens to be the same on both machines. Users wishing to determine whether or not this is the case are advised to experiment.

2. For historical reasons, i4 and m4 type fields in data files on the 6809 Uniflex operating system are not compatible with those on any other machine or operating system, including 68000 Uniflex.

Note that data files must he transferred as binary files and not as text files. It is important to ensure that any file transfer software that you use will transfer all eight bits of each byte and will not modify, add or delete any characters.

Once a data file has been transferred, its index should be rebuilt. The index (.k) files are generally not transferrable between systems. This applies even when moving from Unix version 7 or System 3 up to Unix System V on the same machine. An index may be rebuilt using the program kfri. To ensure that you get the key length and the record length correct, run the program kfdet on the source machine.

If a file cannot be transferred directly for some reason, use the alternative approach outlined on the page 2.

If a file cannot be transferred as outlined on page 1, the following technique may be used:

1. Write a sagerep program to print the content of each record in a format suitable for re-input to a sage program. There are various ways in which this can be done but a simple example is shown below. The method is easily extended if the file contains different record types.

2. Transfer this file to the target machine as a test file. In this case the file transfer software should alter the end of line character(s) if necessary.

3. On the target mechine, write a sage program to input the data. Use newkf to create a new file and then run the sage program, redirecting its input from the transferred text file.

Example sagerep program (pstock.r):

```
!file stock
!final print "e"
        print "i"
        print st_code
        print st_deac
        print st_costpr
        print st_salepr
```

Correasponding sage program (rstock.f)

```
REBUILD STOCK FILE
!file stock
+st_code,,3,20
+st_deac,,4,20
+st_costpr,,5,20
+st_salepr,,6,20

*i=insert
        input st_code - st_salepr
        insert stock
        end

*e=exit
        exit
```

Typical commands to run above programs:

On the source machine: sagerep ptck pvdu >stock.txt

On the target machine: sage rstock <stock.txt

Unix is a trademark of AT & T Bell Laboratories.

Uniflex is a trademark of Technical Systems Consultants.

ESTIMATING SCULPTOR FILE SIZES

Data File

There is no overhead in a aculptor data file. The file size in blocks may be calculated hy multpling the record length by the number of records and then dividing by the block size, which is 512 for most systems and 1024 for Unix System V. The space occupied by deleted records is reused for new insertions.

Index File

The size to which an index file will grov depends on the pattern of insertion and cannot be calculated exactly. Less space is used if records are inserted in a random sequence. The vorst case occurs vhen records are inserted in ascending key sequence. An estimate of the file size may be made as follov.

b = block size (512 most systems, 1024 Unix system V

k = key length in bytes

r = number of records to be inserted

m = maximum number of keys per index block

m = (b - 12) / (k + 5)

For random insertion pattern:

Index file size in blocks = (3 * r) / (2 * m)

For ascending key sequence insertion:

Index file size in blocks = (2 * r) /m

NUMBER OF INDEX LEVELS

Although the performance of a keyed file is elated to the number of index levels, there is little point in attempting to estimate the number of levels that will be built. Each new level exponentially increases the number of records that can be indexed and causes no space overhead. The performance on large files is very impressive.

SCULPTOR VERSION 1.12:4

ENHANCEMENTS TO SAGEREP

1) !cfile [<file_no>] <pathname>

Declares a cross reference file which is to be initially closed. !cfile declarations should precede !xfile declarations. The total number of !file, !cfile and !xfile declarations cannot exceed 16.

2) close <file_no>

Closes and unlocks the specified file. The data in the file's record buffer is still available. The command is ignored if the file is already closed.

3) open <file_no>

Opens the specified file. If the maximum number of files that can be open at one time is exceeded, an error message is generated and sagerep aborts. The command is ignored if the file is already open.

4) abort [<code>]

Causes sagerep to terminate immediately without processing any !final statements. Otherwise similar to exit.

5) display and input are now available as in-line commands. Syntax is the same as !display and !input.

6) A sagerep program with no main statements is now permitted. The program will terminate normally if it reaches the main statement logic.

NOTES

1) Any attempt to access a closed file causes an error message and the program to abort.

2) When a file is closed any locks on that file are removed. If the file is later re-opened, there is no selected record and therefore no write is permitted until a record has been read.

3) Closing and reopening a file leaves the current file position unaltered.

4) On MS DOS, all files accessed by a child task should be closed before doing a chain or exec.

ENHANCEMENTS TO SAGE

1) !cfile [<file_id>] <pathname>

Declares a file which is to be initially closed. !cfile declarations should precede !file declarations. The total number of !cfile and !file declarations cannot exceed 16.

2) close <file_id>

Closes and unlocks the specified file. Th data in the file's record buffer is still available. The command is ignored if the file is already closed.

3) open <file_id>

Opens the specified file. If the maximum number of files that can be open at one time is exceeded, an error message is generated and sage aborts. The command is ignored if the file is already open.

4) preserve <file_id>

Stops the clear command (with no fieldlist) from erasing data in the specified file's record buffer. Effective for the rest of the program. May be used on a file which is open or closed.

SCULPTOR VERSION 1.10:3

The following enhancements, which were introduced in Sculptor version 1.10:3, are particulary useful on the OS9 operating system, but their side effects of enabling files with the same record structure to be easily merged is generally useful.

On OS9 systems, the repeated extension of files can eventually result in the sector map for a file becoming full. When this happens, the file is often logically damaged and is reported as such by kfcheck. The problem can be reduced by formatting the disk with a large cluster size, but this does not guarantee success and wastes space on small files.

A better solution is to pre-extend a file when it is created to its expected maximum size. The program newkf now has this option, and several other programs have been modified to take advantage of it. The enhancements are:

newkf [-rn] <filename> [<filename>]...

Creates pre-extended files with space for 'n' records. Although newkf takes extra time to do this, there is no significant loss of performance when accessing the physically larger file through its index.

kfcopy [-e] [-c] <oldfile> <newfile>
If the [-e] option is specified, <newfile> must exist and may have been pre-extended using newkf. It must have the same key and record length as <oldfile>. Any existing records in <newfile> are retained.

reformat [-e] <oldfile> <newfile>

If the -e option is specified, <newfile> must exist and may have been pre-extended using newdf. Any existing records in <newfile> are retained.

kfri [-c] <filename>

On OS9 only, kfri now automatically pre-extends the new index file to a size computed from the length of the existing data file and key length.

Classified Advertising

Winchester 10 Megabyte Drives

Two (2) 10 Megabyte Hard-Disk Winchester Drives. Working - were removed for upgrade to larger drives.

1 - RMS Model #509 $275.00
1 - Seagate Model #412 $275.00

(615) 842-4600 Tom 9-5 EST.

LSI 68008 CPU card, "C" Compiler and Digital Research CPM/68K $350.

Tano Outpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $595.

MICROKEY Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p. 51 DEC. 84 68' Micro Journal. $1800.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS. $745.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $359.00 ready to run.

S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1290.

1-CDS1 20 Meg Hard Disk System with Controller $1000.

(615) 842-4600 M-F 9 AM to 5 PM EST

***

GIMIX Mainframe, Voice, Music, EPROM, RAM, CPU, Serial, Parallel, Disk Boards, SSSD DRIVES, SOROC, PL9, DYNA C, PASCAL, FORTH, MRP, Display Generator.
(412) 563-5312

***

Programmer wanted for a rapidly growing engineering firm. Company develops and builds hardware and software using Motorola 8 and 16 bit processors. Good background in "C" and assembler required. Starting salary in the lower 20's.
Send resume to:
P.O. Box 8695, Greenville, S.C. 29604

***

Several 2MHZ 64K Static Ram cards, $75 each. Misc. SS50 and SS30 cards. Call with your needs.
(703) 273-6629 evenings.

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4600
Telex 5106006630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: 12758, T2508, T2716, T2516, T2732/2732A, MCM68766/6, T2764/2764A, T2564, T27128/27128A, and T27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL'S intelligent programming (tm) implemented for Intel 2764, 27128 and 27256 devices. Intelligent programming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer. MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l. RETURN ..... to the operating system.
m. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALS.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (%), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND.), (.OR.), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and REALCOM.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Andersons FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPROCS (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V7 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The RSE relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independently or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker
  - Listener
  - System Controller
  - Serial Poll
  - Parallel Poll
  - Group Trigger
  - Single or Dual Primary Address
  - Secondary Address
  - Talk only .. Listen only
- Fully documented with a complete reprint of the KILOMAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-I I/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	S-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order, VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

68' MICRO JOURNAL

Disk-1 Filesort, Minicat, Minicopy, Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.

Disk-2 Diskedit w/ inst.& fixes, Prime. *Prmod, **Snoopy, **Football, **Hexpawn,*Lifetime

Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.

Disk-4 Mailing Program, *Finddat, *Change, *Testdisk.

DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.

Disk-6 **Purchase Order, Index (Disk file indx)

Disk-7 Linking loader, Rload, Darkness

Disk-8 Crtest, Lanplot (May 82)

Disk-9 Dntecopy, Diskfix9 (Aug 82)

Disk-10 Home Accounting (July 82)

Disk-11 Disassembler (June 84)

Disk-12 Modem68 (May 84)

Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help, Date.Txt

Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib

Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo)

Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc

Disk-17 Match Utility, RAT8AS, A Basic Preprocessor

Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray)

Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.

Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C

Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.

Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.

Disk-23 ISAM, Indexed Sequential File Accessing Methods, Condon Nov. 1985. Extensible Table Driven Language Recognition Utility, Anderson March 1986.

Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.

Disk-25 KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.

Disk-26 Compacta UniBoard Review, Code & Diagram. Burlinson March 1986.

NOTE:

This is a reader service **ONLY!** No Warranty is offered or implied, they are **as received** by "**68' Micro Journal**, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

8" Disk $ 14.95 5" Disk $ 12.95

68' Micro Journal

5900 Cassandra Smith Rd. Hixson, Tn. 37343

(615) - 842 -4600

* Indicates 6800

** Indicates BASIC SWTPC or TSC

6809 no Indicator

Foreign Orders Add $4.50 for Surface Mail

or $7.00 for Air Mail

* All Currency in U.S. Dollars

Telex 5106006630

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and – 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density OMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" OSOO Floppies, Cabinet & Power Supply $1698.88
60MB Streamer (UniFIEX-020 only) $2400.00
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) $349.67
#64 Static RAM-64K CMOSw/battery (6809 Only) $398.64
#72 256K CMOS Static RAM w/battery $998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $498.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.46
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS

GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 UniFLEX VM
CPU Included	#05	#05	GMX III	#05	GMX III	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A
25MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included			
FLEX	Included	Included	Included			
GMXBUG Monitor	Included	Included	Included			
Basic 09, Run8 (OS9)	Included	Included	Included			
RMS (OS9)	Included	Included	Included			
DO (OS9)	Included	Included	Included			
VDisk for FLEX	N/A	Included	Included			
RAMDisk for OS9	N/A	$125 option	Included			
O-FLEX	N/A	$250 option	Included			
Support ROM	N/A	N/A	Included			
Hardware CRC	N/A	N/A	Included			

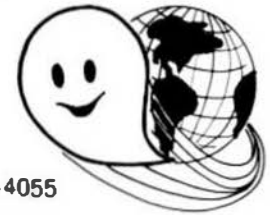
Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

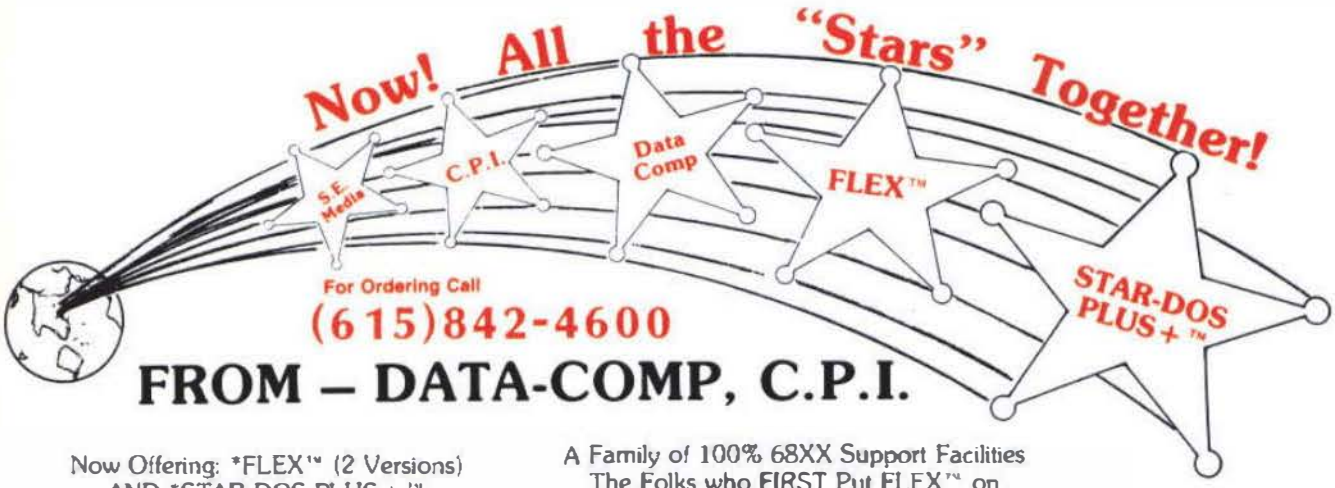
All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX INC.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055





For Ordering Call
(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00
Only $79.95

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&H NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS. $469.95

* Specify What CONTROLLER You Want J&H, or RADIO SHACK

THINLINE DOUBLE SIDED
DOUBLE DENSITY 40 TRACKS $129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&H JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95

CABLES & OTHER INTERFACES
CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

Introducing

S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and confirm a **Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

Thi

Not T

000422 A/E
MR. MICKEY FERGUSON
P. O. BOX 87
KINGSTON SPRINGS TN 37082

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615)842-4607
Telex 5106006630